Raccolta Esercizi di Teoria
(con Soluzioni)

Introduzione:
Sia le prove parziali sia gli appelli ufficiali dell'esame di Programmazione (I Anno della Laurea in Informatica) prevedono una prima parte (circa 15-20 minuti) in cui è richiesto di svolgere alcuni (solitamente 2 o 3) esercizi "teorici"…

Cos'è & cosa c'è:
Ho cercato di raccogliere e mettere insieme testi e soluzioni di vecchi appelli, così che non siano sparsi e difficilmente consultabili in preparazione per l'esame.
Al momento sono ivi contenuti **tutti i testi di teoria che sono riuscito a recuperare dall'Anno Accademico 2011-2012 al corrente (2014-2015 di cui compaiono soltanto i parziali)**; sono esclusi gli esercizi di teoria assegnati durante le lezioni in laboratorio, eccezion fatta per quelli dell'aa 2014-2015.
Sperabilmente questo file verrà espanso ed integrato in futuro.

Le soluzioni:
Quelli presenti li ho svolti tutti e pertanto ho deciso di sistemare quanto ottenuto e creare questo file, di molti sono poi riuscito a recuperare altre precedenti soluzioni che nella maggior parte dei casi provengono da correzioni in aula, e-mail, ricevimenti e quant'altro, in soldoni hanno ricevuto l'approvazione o provengono dal Professore. (Un grazie a tutti coloro che nel tempo le hanno postate).
(in tal caso sono marcate come "Soluzione by Filè", altrimenti sono semplicemente "NON verificate").

Il libro:
Ove ci siano riferimenti ad un libro di testo s'intende:
"Programmazione consapevole (semplice è bello)" di G. Filè, Libreria Progetto Padova, 2° o 3° Ed.

Disclaimer:
Quelle qui presenti non intendono essere soluzioni da 30L per gli esami ma piuttosto una raccolta di "Tracce di Soluzioni"; chi interessato dovrebbe svolgere gli esercizi di proprio pugno ed usare le soluzioni come primo feedback e confronto o come input per un successivo approfondimento dell'argomento in caso di dubbi.
Si declina ogni tipo di responsabilità, conseguenza e quant'altro.
In caso di contraddizione tra questo testo ed una qualunque altra fonte (attendibile) fa fede l'altra fonte. Chi ne fa uso dichiara implicitamente di essere a conoscenza di quanto sopra e di assumersene i rischi.

NB: qui nessuno è pagato per far nulla, è tutto un contribuito gratuito e volontario alla "comunità", l'unica cosa richiesta in cambio è (possibilmente) un grazie, e rispetto del lavoro, tempo ed impegno degli altri. **Se puoi, contribuisci anche tu, estendi, correggi, modifica, aggiorna questo contenuto e fai si che chi viene dopo di te lo trovi migliore di come l'hai trovato, se ti è stato utile lo sarà anche al tuo prossimo.**

Quanto qui presente è soggetto ai naturali errori di trascrizione, battitura, interpretazione, svista e quant'altro, ma soprattutto (i più gravi) i classici "errori di sbaglio" cioè i veri e propri errori; se ne trovate segnalateli o (ancor meglio) correggeteli. Se avete dubbi su una soluzione proposta: discutetene, chiedete delucidazioni, avete un libro, dei compagni, dei tutor ed un professore: usateli. Una volta raggiunto un verdetto eventualmente aggiornate il presente file così il vostro prossimo avrà meno grattacapi.

Grazie a tutti, LM

Un piccolo consiglio:

Prima di iniziare a svolgere gli esercizi di teoria (anche quelli presenti in questo documento) è consigliato:
Leggere il Capitolo 9 del libro di testo.
Studiare e svolgere personalmente gli esempi e gli esercizi delle slide delle lezioni, nello specifico tutto il contenuto di "Esercizi sulle Funzioni: passaggio dei parametri, restituzione dei risultati e invocazione"; sono molto utili e di difficoltà crescente, sarebbe meglio li svolgeste e poi li confrontaste con gli appunti della lezione in cui sono stati spiegati in aula. Fateli ma soprattutto capiteli nel dettaglio, ogni simbolo, virgola, o cosa, allora sarete ad un buon punto.

Dettagli, Legenda e Notazione:

La notazione in uso è quella adottata dal professore.
In particolare si noti che:
- "L è una lista corretta" o "lista(L) corretta" sono equivalenti.
- "R è albero corretto" o "albero(R) corretto" sono equivalenti.
- Con lista(L) s'intende una lista concatenata di nodi in cui L è un puntatore al primo nodo.
- Con albero(R) s'intende un albero (solitamente binario) ed R punta al suo vertice (root).
- Con albero e/o lista "corretta/o" s'intende sempre (anche) possibilmente vuota/o (a volte è esplicitato, altre no, nel caso si voglia imporre L!=0 e/o R!=0 va sempre esplicitato "non vuoto").
- "vX" significa "vecchio X", inteso X prima che avvenga l'iterazione corrente (ad esempio vL è vecchia lista, vR è vecchio albero, vdimP è vecchio dimP).
- che un (o più) valore sia "definito" (a volte "def.") significa che esso è del tipo corretto e non è indefinito (è stato inizializzato correttamente, non è un valore sconosciuto/randomico/residuo di RAM).

Disegni, Immagini e Schemi

Per quanto "rudimentali" sono stati aggiunti perché richiesti e utili, nella speranza che siano per lo meno comprensibili, si ricorda che:
- la RAM è sempre una sola, anche se rappresentata in due pezzi separati per facilitare il disegno dei puntatori.
- i "rettangoli neri" della RAM sono lo spazio allocato per la variabile indicata a sinistra (o a destra) di ogni singolo rettangolo (contengono il corretto numero di byte necessari siano essi 4, 8, o altro); rettangoli raggruppati rappresentano array.
- in caso di "aliasing" (ovvero nel caso si dichiarasse un "riferimento" ad una variabile) il nuovo/secondo nome sarà elencato vicino all'originale dato che condividono lo stesso R- ed L-valore.
- le "frecce" (tipicamente rosse) sono "l'espressione visiva" dei puntatori, esse partono dall'interno del rettangolo che rappresenta la variabile puntatore ed indicano (puntano) ad un altro rettangolo, cioè a quello cui si riferiscono (ottenibile dereferenziando una volta).
- quando qualcosa è "barrato" in grigio è segno che prima c'era ed ora (allo stato finale) è stato cancellato/rimpiazzato (ad esempio un puntatore è stato spostato, o un valore sovrascritto).
- un'espressione in grigio vicino ad una freccia grigia tipicamente rappresenta l'R-val di una variabile puntatore restituito da una funzione, la destinazione della freccia indica dove punta.
- i nodi di un albero sono rappresentati da dei cerchi, i puntatori ai discendenti da degli archi che collegano due cerchi, un cerchio senza archi nella parte bassa (in uscita) non ha figli (è una foglia), un cerchio senza archi nella parte alta (in entrata) è la radice/root ed è tipicamente puntata da R; distinguere quale sia il figlio sinistro e quale il destro dovrebbe essere visivamente immediato.

Elenco del Contenuto:

I Parziale (II semestre) (aa 2014-2015)
22 aprile 2015

(1a) Dire se il seguente programma è corretto o meno. Spiegare la propria risposta.
Si consiglia di disegnare uno schema delle relazione tra le variabili del programma durante l'esecuzione.

```
int** F(int*x){int**p=&x; (*x)++; return x;}
main(){int a=2,*q=&a; **F(q)=4; cout<<a<<endl;}
```

(1b) Dire se il seguente programma è corretto o meno. Spiegare la propria risposta.
Si consiglia di disegnare uno schema delle relazione tra le variabili del programma durante l'esecuzione.

```
int** F(int*&x){int**p=&x; (*x)++; return x;}
main(){int a=2,*q=&a; **F(q)=4; cout<<a<<endl;}
```

(2a) Dato int T[3][5][6], dire che tipo ha l'espressione: *(T[8]-2)

(2b) Dato int T[3][5][6], dire che tipo ha l'espressione: ((*T)+8)[-2]

(3a) Cosa stampa il seguente programma?

```
int x=0;
for(int i=0; i<10; i++)
{ cout<<x<<endl;
  if(x>0)
    break;
  else
    continue;
  x++;
}
```

(3b) Cosa stampa il seguente programma?

```
int x=1;
for(int i=0; i<10; i++)
{ cout<<x<<endl;
  if(x>0)
    continue;
  else
    break;
  x=0;
}
```

Nota:
l'esame era diviso in due turni, gli esercizi (Xa) sono quelli del primo turno, mentre (Xb) quelli del secondo.

SOLUZIONE I Parziale (II semestre) (aa 2014-2015)
22 aprile 2015
(soluzione by Filè)

(1a) e (1b)
In entrambi la funzione F contiene un errore di tipo:
deve restituire un int**, ma return x; restituisce un valore int*.
Quindi il programma non compila neppure.
Senza la presenza di suddetto errore, il main rispetterebbe le concordanze di tipi.

(2a)
A T vengono applicate 2 *, una esplicita e l'altra contenuta nel subscripting in T[8]=*(T+8), visto che il tipo di T è: int (*)[5][6], applicando 2 stelle (dereferenziazioni) si ottiene un int*.

(2b)
Similmente a quanto visto per la soluzione dell'esercizio (2a), anche qui si applicano 2 stelle a T e quindi il tipo finale è int*.

(3a)
visto che x è 0, la condizione esegue sempre il ramo else e quindi esegue il "continue" che causa il salto del x++. Quindi il ciclo viene eseguito 10 volte e stampa ogni volta 0.

(3b)
x è 1 all'inizio e quindi il condizionale entra nel ramo then e quindi esegue il "continue" che salta l'assegnazione x=0 che non viene mai fatta. Quindi il ciclo esegue 10 volte e stampa 10 volte 1.

Nota:
fatto salvo che quanto qui presente è la soluzione ufficiale presentata dal professore, al seguente link potete trovare una trattazione più specifica delle stesse domande, con commenti ed errori precisi per ogni esercizio: http://pastebin.com/t0BrLhZU

II Parziale (II semestre) (aa 2014-2015)
09 giugno 2015

(1) Cercate di scrivere una PRE ed una POST sensate per il seguente programma ricorsivo:

```
int M(nodo *R, int *P, int dimP)
{
if(!dimP || !R)
  return 0;
int z=0;
if(R->info==*P)
{ z=1; P++; dimP--; }
int a=M(R->left, P, dimP);
int b=M(R->right, P, dimP);
if(a>=b)
  return a+z;
else
  return b+z;
}
```

(2) Si supponga che due persone vogliano realizzare contemporaneamente le due parti di un programma: f1.cpp ed f2.cpp.
Una volta accordatisi risulta che f2.cpp userà una funzione (fun1) di f1.cpp e f1.cpp farà uso di una struttura (struct T) di f2.cpp.
Rispondere sinteticamente a tutti i seguenti punti:
a) è possibile per i due programmatori compilare solamente la loro parte?
b) quanti main() ci possono essere in totale tra f1.cpp e f2.cpp?
c) descrivere in che modo è possibile per f2.cpp usare fun1 e per f1.cpp usare la struttura.

Nota:
Il testo degli esercizi non è (ancora) stato rilasciato ufficialmente, è pertanto stato “ricostruito”; principalmente per quanto riguarda l’esercizio (2), è possibile che non sia esattamente identico a quello svolto in aula, ma sostanzialmente le domande/argomenti sono quelle.

SOLUZIONE II Parziale (II semestre) (aa 2014-2015)
09 giugno 2015
(soluzione NON verificata)

(1)
PRE=(albero(R) corretto, P[0..dimP-1] def, dimP>=0, albero(vR)=albero(R)).
POST=(detto lung l'intero ritornato, allora il match (possibilmente non contiguo) di lunghezza massima di P[0..dimP-1] su albero(R) ha lunghezza lung e albero(R)==albero(vR)).
Nota: il match è cercato lungo tutti i cammini di albero(R), non sono considerate come parti di un eventuale stesso match "pezzi" che risiedono su cammini diversi; in altre parole di volta in volta si sceglie il sottoalbero che contiene più match singoli e pertanto non sono considerati parte dello stesso match (ad esempio) il nodo più in basso a sinistra ed il nodo più in basso a destra.

(2)
(la soluzione seguente è stringata, e cerca di coprire tutti i punti salienti della risposta "completa"; per una trattazione chiara e dettagliata dell'argomento vedere: "9.7 Compilazione separata e namespace" del libro di testo).
a) sì, ed è comodo poterlo fare per poter individuare al più presto eventuali errori, warning ed indicazioni del compilatore; le due parti (f1.cpp) e (f2.cpp) prendono il nome di "unità di compilazione", il compilatore permette la compilazione di una singola unità con il comando "g++ -c f1.cpp" (o f2.cpp), il risultato sarà un file compilato f1.o (o f2.o) (sarà poi il modulo "linker" del compilatore a "mettere insieme" tutti i file ".o").
b) quando da f1.cpp e f2.cpp si vuole ricavare un eseguibile (.exe/.out) allora è necessario che ci sia uno e un solo main() (in uno tra f1.cpp/f1.o e f2.cpp/f2.o), altrimenti la fase di "linking" (tipicamente realizzata tramite l'uso di "makefile", vedi libro di testo) produrrà un errore.
c) il modo migliore sarebbe tramite i file header (.h) (vedi libro di testo); in alternativa (per semplificare):
- f1.cpp deve contenere un'esatta copia della dichiarazione del tipo strutturato T così come appare anche in f2.cpp (ciò viola la regola della dichiarazione singola, ma è accettato dal C++ dato che queste dichiarazioni di tipo non richiedono l'allocazione di memoria e quindi non introducono tutti i problemi tipicamente legati alle dichiarazioni multiple, per ulteriori dettagli vedere libro di testo).
- f2.cpp deve contenere solo il prototipo della funzione fun1 (prima di invocarla), tale prototipo è sufficiente al compilatore per fare il "controllo sui tipi".

II Appello (aa 2013-2014)
31 marzo 2014

(1) Data la seguente funzione ricorsiva, inserire appropriate PRE e POST

```
//PRE= ??
int F(Nodo* a) {
if(!a) return 0;
if (a->left) return 1+F(a->left);
if(a->left || a->right) return 2+F(a->left)+F(a->right);
return 3+F(a->left)+F(a->right);
} //POST=??
```

(2a) Considerare il seguente programma e dite se è corretto o no. Se pensate che sia corretto, spiegate i passi dell'esecuzione. Se pensate che sia sbagliato, spiegate con precisione perché.

```
int* f(int **p){int b=3,*x=&b; **p=*x; *x=**p; return x; }
main() {int y=5, b=2,*q=&b; *f(&q)=y*2; cout<<y<<b<<*q;}
```

(2b) Considerare il seguente programma e dite se è corretto o no. Se pensate che sia corretto, spiegate i passi dell'esecuzione. Se pensate che sia sbagliato, spiegate con precisione perché.

```
int* f(int **p){int b=3,*x=&b; **p=*x; x=*p; return x; }
main() {int y=5, b=2,*q=&b; *f(&q)=y*2; cout<<y<<b<<*q;}
```

3) Un programma può essere scritto su più file. Ogni file può venire compilato da solo, ma alla fine, tutti i file dovranno venire compilati assieme. Supponiamo di avere un programma scritto su due file e supponiamo che le funzioni scritte su ciascuno dei due file usino uno stesso tipo struttura P ed una stessa funzione f che sono come segue:

```
struct P{int a,b; P* next;};
P* f(P* x){.....}
```

Esattamente cosa di P e cosa di f deve essere scritto su ciascun file? Spiegare brevemente la risposta.

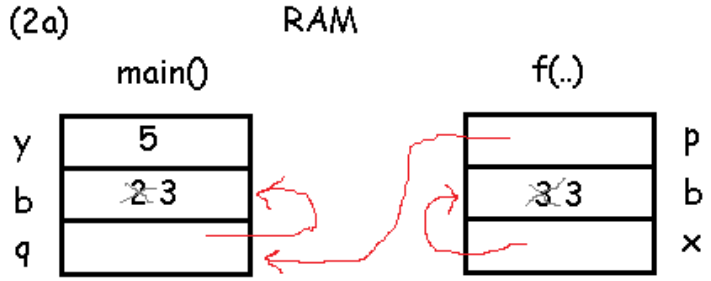
Nota:
gli esercizi (1) e (3) sono uguali sia per il turno I sia per il turno II dell'esame,
l'esercizio (2) invece ha due versioni (qui riportate come (2a) e (2b)).

SOLUZIONE II Appello (aa 2013-2014)
31 marzo 2014
(soluzione by Filè)

(1)
PRE=(a è albero corretto).
POST=(viene considerato il cammino dalla radice ad una foglia che è il più a sinistra possibile, cioè ogni volta che c'è il figlio sinistro si va a sinistra e solo se non c'è si va a destra, e ad ogni nodo di questo cammino si associa un intero in questo modo:
-se si va a sinistra, 1;
-se si va a destra, 2;
-per la foglia finale, 3.
La funzione restituisce la somma di tutti gli interi del cammino).

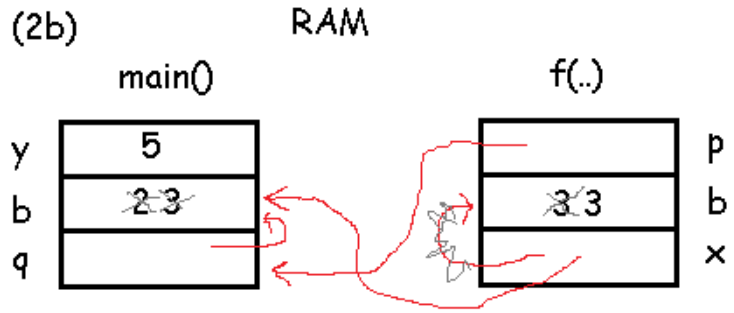
(2a)

il main passa l'L-valore di q che diventa l'R-valore di p. x punta a b locale alla funzione f, e **p è invece il b del main che riceve l'R-valore del b di f, poi *x==**p non fa nulla perché significa che il b di f riceve l'R-valore del b del main (che ha appena ricevuto l'R-valore del b di f!). Infine si ritornal'R-valore di x che punta al b di f! Quindi è **un dangling pointer** e *f(&q) lo dereferenzia. Errore di tipo: si accede una variabile (b di f) che è stata deallocata.



(2b)

In questa versione in f c'è x=*p che cambia completamente le cose rispetto al precedente esercizio. Infatti x riceve *p cioè l'R-valore di q che punta a b del main, quindi f ritorna l'indirizzo di b del main e la sua dereferenziazione non causa errori: b del main riceve 10 e quindi la stampa produce, 5, 10, 10.
Il programma è corretto.



(3)
un file deve avere l'intera definizione di f, mentre l'altro deve contenere solo il suo prototipo. Entrambi i file devono contenere l'intera definizione di P (esattamente la stessa). Per spiegazioni leggere il capitolo 9.

III Appello (aa 2013-2014)
03 luglio 2014

(1) Considerate il seguente programma. In caso pensiate sia corretto, spiegate cosa calcola e cosa stampa. Se pensate sia sbagliato, spiegate cosa c'è che non va. È richiesto il disegno dello stato della memoria durante l'esecuzione. Senza disegno non si riceve alcun punto.

```
int** f(int * & p){int**x=&p; (*((*x)+1))++ ; p--; return x; }
main() {int b[]={2,3,4,5},*q=b+1; **f(q)=*q; cout<<b[0]<<b[1]<<b[2]<<b[3];}
```

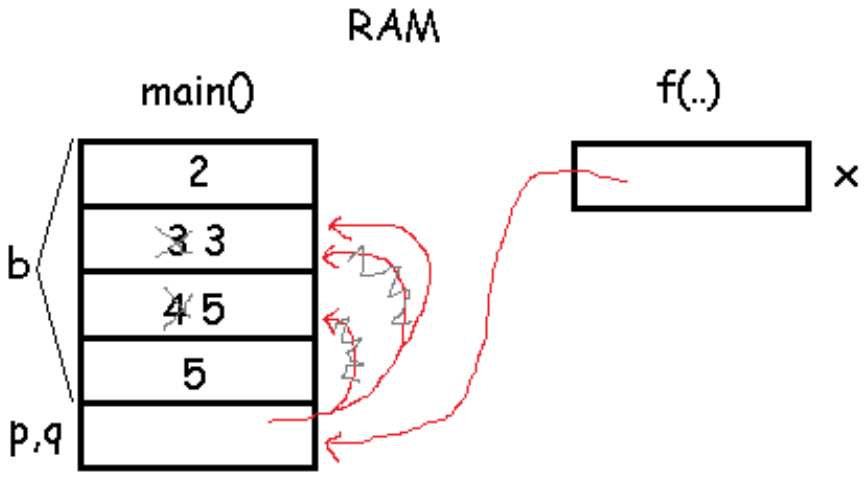
(2) Si dia PRE e POST alle seguenti 2 funzioni ricorsive:

```
int F(nodo*L)
{if(!L) return 0;
int x= G(L);
int y=F(L->next)
if(x>=y)
  return x;
else
  return y;
}
```

```
int G(nodo*L)
{
if(!L) return 0;
if(!L->next)
  return 1;
if(L->info<=L->next->info)
  return 1+G(L->next);
else
  return 1;
}
```

SOLUZIONE III Appello (aa 2013-2014)
03 luglio 2014
(soluzione NON verificata)

(1)
Si noti che l'istruzione "**f(q)=*q;" è ambigua, dato che la funzione fa side-effect sul valore puntato da "q" ed il C++ non specifica quale delle due parti viene eseguita per prima, tuttavia in questo caso non cambia il risultato a prescindere che sia valutata prima la parte destra o la sinistra. Stampa: 2355.
(vedere schema)



(2)
PRE_G=(L è lista corretta).
POST_G=(conta e ritorna i nodi consecutivi in ordine di campo info crescente).
PRE_F=(L è lista corretta).
POST_F(ritorna il miglior risultato di G, cioè il numero di nodi dello "spezzone" più lungo in L in cui i nodi consecutivi sono ordinati per campo info crescente).

IV Appello (aa 2013-2014)
18 luglio 2014

(1) Considerate il seguente programma. In caso pensiate sia corretto, spiegate cosa calcola e stampa. Se pensate sia sbagliato, spiegate cosa c'è che non va. È richiesto il disegno dello stato della memoria durante l'esecuzione. Senza disegno non si riceve alcun punto.

```
int*& f(int * & p){int*& x=p; ++x; ++p; return x; }
main() {int b[]={2,3,4,5},*q=b; f(q)=b; cout<<*q<<b[0]<<b[1]<<b[2]<<b[3];}
```

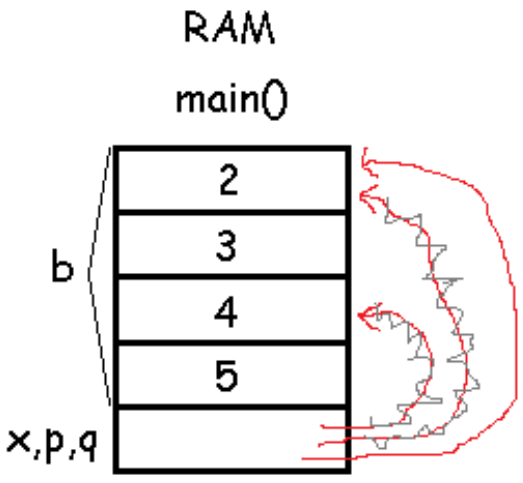
(2) Si trovino PRE e POST appropriate per la seguente funzione :

```
nodo* F(nodo*L)
{
if(!L->next) return L;
nodo*x=L->next;
L->next=x->next;
x->next=F(L);
return x;
}
```

(3) Si consideri la seguente dichiarazione: char X[3][4][4][5]
a) che tipo ha (*X)[-3] ?
b) (assumendo che X abbia valore X) che tipo e che valore ha *(X[-2])+1 ?

SOLUZIONE IV Appello (aa 2013-2014)
18 luglio 2014
(soluzione NON verificata)

(1)
È corretto e stampa: 22345
(vedere schema)



(2)
PRE=(L è lista corretta e non vuota, sia vL=L).
POST=(se vL=a@L' ritorna lista X corretta tale che X=L'@a, cioè sposta il primo nodo alla fine).

(3)
(mancante)

I Appello (aa 2012-2013)
21 marzo 2013

(1) Supponete di avere un programma scritto in parte sul file F1.cpp ed in parte sul file F2.cpp. È possibile che nel blocco globale di F1.cpp ci sia la dichiarazione int pippo; e in quello di F2.cpp ci sia la dichiarazione double pippo=3.14;? Spiegate brevemente la vostra risposta.

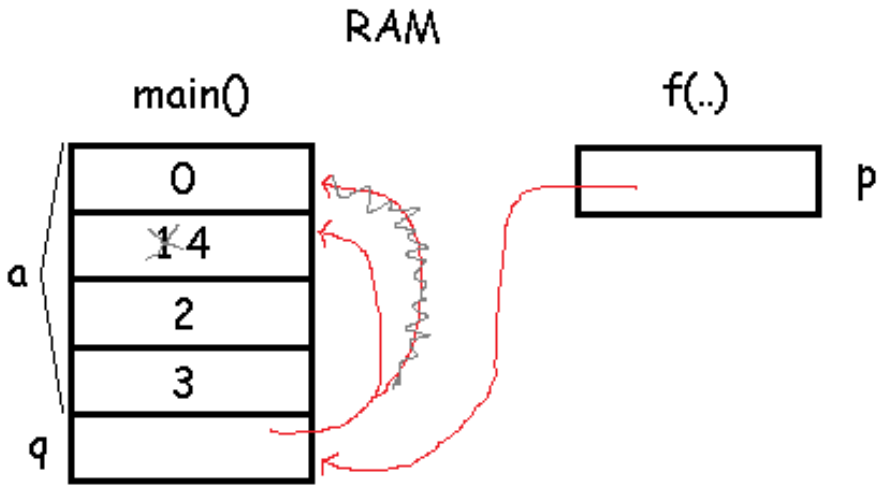
2) Considerate il seguente programma:
int ** F(int** p){(*p)++; return p;}
main(){int a[]={0,1,2,3}, *q=&(*a); **F(&q)=a[3]+1; cout<<q[0]<<q[1]<<q[2]<<a[3];}
Se pensate che il programma sia corretto spiegate cosa succede durante la sua esecuzione e cosa stampa, oppure, se pensate che contenga errori, spiegateli.

3) Data la seguente dichiarazione char X[10][5][10][5]; che tipo e che valore ha l'espressione seguente?
(X[3][-5]) – 4
Si assuma che l'R-valore di X sia X.

SOLUZIONE I Appello (aa 2012-2013)
21 marzo 2013
(soluzione NON verificata)

(1)
No, le compilazioni separate funzionerebbero senza problemi, ma nella fase di "link" ci sarebbe un errore di doppia dichiarazione della variabile pippo (una int e l'altra double). Per spiegazioni leggere il capitolo 9.

(2)
È corretto e stampa: 4233
Nota: prestare attenzione all'ultima istruzione di stampa, usa il puntatore "p" ed anche l'array "a".
(vedere schema)



(3)
(mancante)

II Appello (aa 2012-2013)
28 marzo 2013

(1a) Dire se il seguente programma è corretto o meno e perché.
In caso pensiate sia corretto dite cosa stampa e perché.

```
const int x=99, *p=&x;
int* q= const_cast<int*>(p);
(*q)++;
cout<<x<<*q<<*p<<endl;
```

(1b) Dire se il seguente programma è corretto o meno e perché.
In caso pensiate sia corretto dite cosa stampa e perché.

```
int x=99, *const p=&x;
int * const * q=&p;
(**q)++;
cout<<x<<**q<<*p<<endl;
```

(2a) Cercate di scrivere una PRE ed una POST sensate per il seguente programma ricorsivo:

```
int F(nodo* R){
if(! R) return 0;
if(R->left && R->right) return F(R->left) + F(R->right);
if(R->left|| R->right) return 1+F(R->left) + F(R->right);
return 0;
}
```

(2b) Dotare la seguente funzione ricorsiva di adeguate pre- e post-condizioni:

```
int F(nodo*R) {
if(!R->left && !R->right) return 1;
if(!R->left) return F(R->right);
if(!R->right) return F(R->left);
return F(R->left)+F(R->right);
}
```

(3) Data la seguente dichiarazione char X[10][4][10][8]; che tipo e che valore ha l'espressione seguente?
(X+2)[-5]- 4
Si assuma che l'R-valore di X sia X.

Nota:
non sono sicuro che gli esercizi (1b) e (2b) facessero parte dell'esame (probabilmente erano quelli del secondo turno di quell'esame) comunque data la loro somiglianza con gli esercizi (1a) e (2a) li ho messi qui con relative soluzioni.

SOLUZIONE II Appello (aa 2012-2013)
28 marzo 2013
(soluzione NON verificata)

(1a)
È corretto: x costante intera, p è un puntatore ad x (p ha tipo const int *), q è un nuovo puntatore a int (non const) e viene inizializzato con l'R-val di p previa un cast che permette l'assegnazione forzata di un punt a const int (p) ad un punt a int (q); ora tramite q si può cambiare x perché q è un puntatore normale (senza nessun const).
(molto probabilmente) stampa: 99100100 (cioè 99,100,100).
Questo perché:
esiste una sola x, ma spesso i compilatori sostituiscono nel codice le occorrenze dei nomi delle costanti (qui x) con il loro valore iniziale (qui 99). Pertanto le modifiche (tramite q) ad x avvengono affettivamente, ma ovunque ci fosse stato x a compile-time è avvenuta una sostituzione, comunque sia tramite q, sia tramite p si stamperà il nuovo valore di x (100).

(1b)
È corretto, segue breve spiegazione:
x è una variabile di tipo int (non costante).
p è un puntatore costante ad x (non costante), praticamente non si può "spostare" il puntatore (o meglio: non si può cambiare l'R-valore della variabile p) e quindi operazioni come p++; vengono rigettate dal compilatore; questo però non influenza minimamente x, per cui tramite p si può ottenere e modificare x (ad esempio istruzioni come: (*p)++; sono legittime).
q è un puntatore a puntatore costante (p), a sua volta p è un puntatore (costante) ad int (x);
q non ha alcun vincolo e si può quindi fare: q++; tuttavia punta ad un "oggetto costante" (in questo caso al puntatore p) che quindi non può essere modificato, infatti (*q)++; è segnalato dal compilatore come un errore perché appunto si tenta di cambiare l'R-val di p; infine tramite q possiamo ottenere p (non modificarlo) e tramite p ottenere x e modificarlo: (**q)++; è un'operazione legittima che porta il valore di x da 99 a 100.
Stamperà dunque: 100100100 (cioè: 100,100,100).

(2a)
PRE=(R è albero corretto).
POST=(restituisce il numero dei nodi dell'albero che hanno un solo figlio (o sinistro o destro)).

(2b)
PRE=(R è albero corretto e non vuoto)
POST=(restituisce il numero delle foglie dell'albero).

(3)
(mancante)

II Parziale (aa 2011-2012)
14 marzo 2012

(1) Scrivere PRE e POST opportune per la seguente funzione ricorsiva:

```
//PRE= ??
bool H(nodo*L, int & k)
{ if(!L) { k=1; return false; }
if(H(L->next, k)) {k=k+1; return false;}
else return true;
}
```

POST= ??

SOLUZIONE II Parziale (aa 2011-2012)
14 marzo 2012
(soluzione by Filè)

(1)
PRE=(L è lista corretta, k è definito).
POST=(detta lung la lunghezza (numero di nodi) di L, allora:
se lung pari => false e k=lung/2+1
se lung dispari => true e k=(lung-1)/2 +1).
Nota: praticamente se il numero di nodi di lista L è dispari, restituisce true e k è il numero di nodi dispari presenti; se invece lista L ha un numero di nodi pari, allora restituisce false e k è il numero di nodi pari+1.

II Appello (aa 2011-2012)
29 marzo 2012

(1) Data la dichiarazione char X[4][8][10], qual è il tipo di (*X)[-2] e quale dimensione ha l'oggetto puntato da questo puntatore? Inoltre che valore ha l'espressione (*X)[-2] rispetto al valore di X?

(2)
Spiegare cos'è l'overloading (sovraccaricamento) ed i principi dell'overloading resolution. (max 4 righe).

SOLUZIONE II Appello (aa 2011-2012)
29 marzo 2012
(soluzione NON verificata)

(1)
(mancante)

(2)
Per spiegazioni leggere il capitolo 9 (in particolare la sezione 9.8).

I Appello (aa 2011-2012)
20 marzo 2012

(1) Data la seguente funzione ricorsiva, inserire appropriate PRE e POST

```
//PRE=??
int F(nodo *a){
if(!a) return 0;
if(a->left && a->right) return F(a->right)+F(a->left);
return 1+F(a->left)+ F(a->right);
}
//POST=??
```

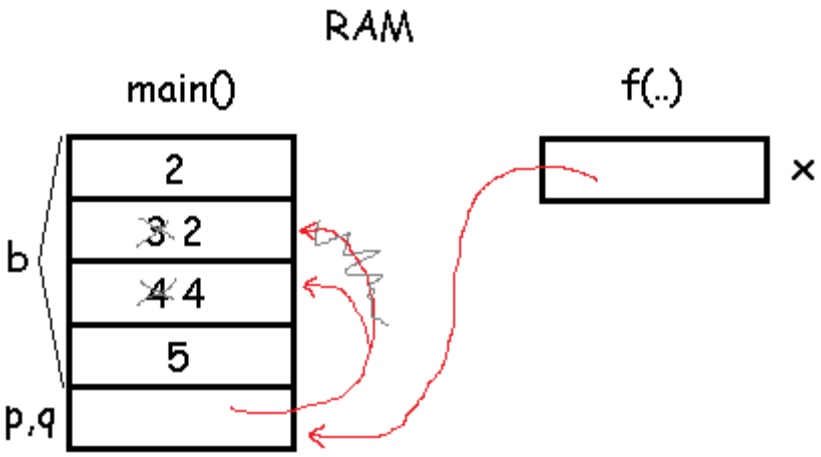
(2) Considerate il seguente programma. In caso pensiate sia corretto, spiegate cosa calcola e cosa stampa. Se pensate sia sbagliato, spiegate in dettaglio il motivo.

```
int ** f(int *&p){int**x=&p; p[0]--; p++; return x;}
main() {int b[]={2,3,4,5}, *q=b+1; **f(q)=*q; cout<<b[0]<<b[1]<<b[2]<<b[3];}
```

SOLUZIONE I Appello (aa 2011-2012)
20 marzo 2012
(soluzione NON verificata)

(1)
PRE=(a è albero corretto).
POST=(restituisce il numero di nodi dell'albero che hanno al più un figlio (quindi quelli con 1 solo figlio e le foglie).

(2)
Potremmo dire che è corretto, ma vale la pena notare che:
Per quanto ci sia concordanza di tipi e quindi non occorrano errori di compilazione e non ci sia nessun problema di dangling pointer, la chiamata **f(q)=*q; è ambigua; questo perché: la parte sinistra dell'assegnazione opera un side-effect sull'R-valore di *q che "serve" poi a destra dell'assegnazione, questo implica quindi che il risultato dipende dall'ordine con cui vengono valutate le due parti (lo standard C++ non lo specifica, ma normalmente viene valutata prima la chiamata di funzione).
Pertanto, se prima viene valutata la parte sinistra stamperà: 2245
Se invece viene valutata la parte destra per prima stamperà: 2225
(vedere schema che si riferisce alla prima delle due stampe)

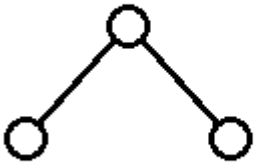


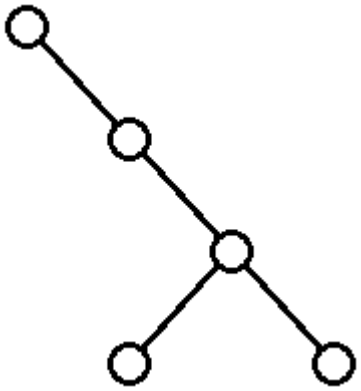
III Appello (aa 2011-2012)
19 giugno 2012

(1) Data la seguente funzione ricorsiva:

```
inf F(nodo* a) {
if(!a) return false;
if(a->left && a->right) return F(a->right) || F(a->left);
return !( F(a->left) || F(a->right));
}
```

Dire cosa calcola F invocate sulle radici dei due alberi raffigurati. Sulla base di questi valori, specificare un'appropriata coppia di PRE e POST per la funzione F.





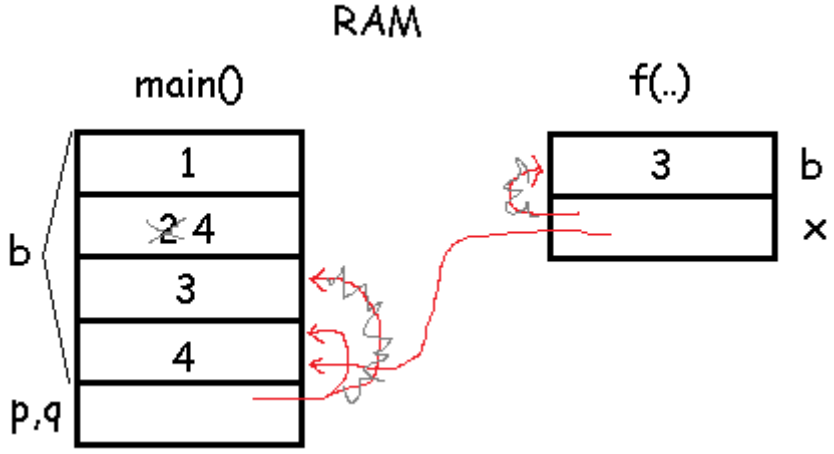
(2) Considerate il seguente programma. In caso pensiate sia corretto, spiegate cosa calcola e cosa stampa. Se pensate sia sbagliato, spiegate in dettaglio gli errori. In ogni caso, per spiegare le vostre conclusioni, disegnate uno schema dettagliato che mostri la relazione tra le variabili e i puntatori in gioco.

```
int *f(int *&p){int b=3, *x=&b; x=p+1; p++; return x-2;}
main() {int b[]={1,2,3,4}, *q=b+2; *f(q)=*q; cout<<b[0]<<b[1]<<b[2]<<b[3];}
```

SOLUZIONE III Appello (aa 2011-2012)
19 giugno 2012
(soluzione by Filè)

(1)
L'albero raffigurato a sinistra ritorna: 1 (true)
L'albero raffigurato a destra ritorna: 1 (true)
Nota0: se l'albero di destra avesse avuto un totale di 5 nodi (anziché 3) sulla "diagonale lunga" allora avrebbe restituito: 0 (false); il motivo sembra essere legato al numero di nodi con esattamente un solo figlio, o meglio se questi (i nodi, non i figli) sono presenti in numero pari o in numero dispari, e le fogli non contano
PREMESSA:
un cammino è detto "stabile" se un nodo x (con 2 figli) del cammino ha valore false (il che significa che il suo figlio che appartiene al cammino ha anche lui valore false), allora x nell'albero deve avere valore false. Il che significa che tutti i cammini stabili che partono da x e arrivano ad una foglia danno valore false per x. Insomma i cammini stabili danno un valore di verità ai loro nodi che è uguale a quello che essi hanno nell'albero.
PRE=(a è albero corretto).
POST=(la radice dell'albero è true sse esiste un cammino "stabile" dalla radice ad una foglia con un numero pari di nodi con un solo figlio).
Nota1: il valore ritornato è 1 o 0 di tipo int, grazie ad una conversione (promozione) automatica bool->int.
Nota2: la premessa non è molto chiara (?), ma dal codice, in pratica (se consideriamo lo zero come un numero pari) sembra che:
- se ritorna 1 (true) allora ci sono un numero pari di nodi (interni) dell'albero con esattamente un solo figlio;
- se ritorna 0 (false) e a!=0 (l'albero iniziale non è vuoto) allora ci sono un numero dispari di nodi (interni) con esattamente un solo figlio;
- altrimenti (cioè se ritorna 0 e a==0) allora l'albero iniziale è vuoto.

(2)
Nella funzione f: p è alias di q del main. Potremmo dire che è corretto, ma vale la pena notare che:
per quanto ci sia concordanza di tipi e quindi non occorrano errori di compilazione e non ci sia nessun problema di dangling pointer, la chiamata *f(q)=*q; è ambigua; questo perché: la parte sinistra dell'assegnazione opera un side-effect sull'R-valore di *q che "serve" poi a destra dell'assegnazione, questo implica quindi che il risultato dipende dall'ordine con cui vengono valutate le due parti (lo standard C++ non lo specifica, ma normalmente viene valutata prima la chiamata di funzione).
Pertanto, se prima viene valutata la parte sinistra stamperà: 1434
Se invece viene valutata la parte destra per prima stamperà: 1334
(vedere schema che si riferisce alla prima delle due stampe)

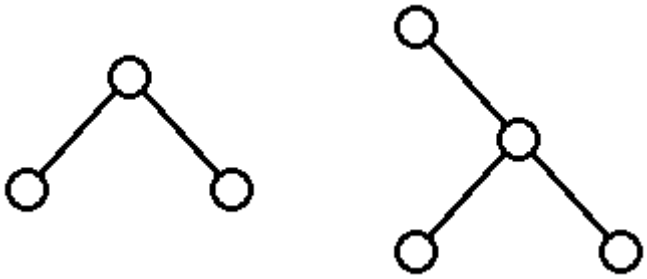


IV Appello (aa 2011-2012)
16 luglio 2012

(1) Si consideri il seguente programma ricorsivo che riceve in input un albero binario (corretto):

```
bool F(nodo* a) {
if(!a) return true;
if(F(a->left) != F(a->right))
  return true;
else
  return false;
}
```

Dire quali sono i valori di verità che F calcola per i seguenti due alberi, e sulla base dei valori di F per questi due alberi, cercate di trovare una POST-condizione per tutti i possibili alberi di input.



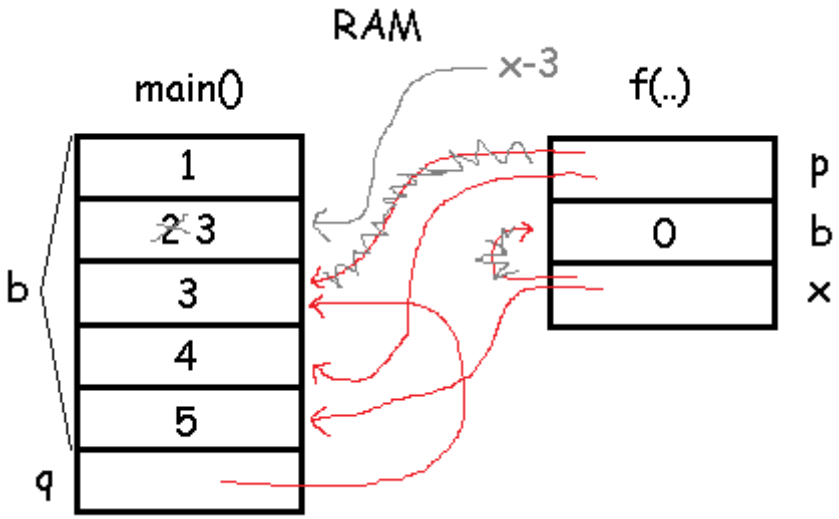
(2) Considerate il seguente programma. In caso pensiate sia corretto, spiegate cosa calcola e cosa stampa. Se pensate sia sbagliato, spiegate in dettaglio gli errori. In ogni caso, per spiegare le vostre conclusioni, disegnate uno schema dettagliato che mostri la relazione tra le variabili e i puntatori in gioco. In nessun caso verranno assegnati punti per risposte senza spiegazioni.

```
int *f(int *p){int b=0, *x=&b; x=p+2; p++; return x-3;}
main(){int b[]={1,2,3,4,5},*q=b+2; *f(q)=*q; cout<<b[0]<<b[1]<<b[2]<<b[3]<<b[4];}
```

SOLUZIONE IV Appello (aa 2011-2012)
16 luglio 2012
(soluzione by Filè)

(1)
L'albero raffigurato a sinistra ritorna: false
L'albero raffigurato a destra ritorna: true
PRE è la prima riga del testo dell'esercizio (l'input è un albero binario corretto).
POST=(ritorna false sse l'albero ha un numero dispari di nodi (comprese le foglie);
ritorna true sse l'albero ha un numero pari di nodi (comprese le foglie)).

(2)
C'è concordanza di tipi e quindi non occorrano errori di compilazione e non c'è nessun problema di dangling pointer, p di f è una "copia" di q del main (passaggio del parametro per valore) la chiamata *f(q)=*q; non è ambigua; questo perché: la parte sinistra dell'assegnazione non fa side-effect sull'R-valore di *q che "serve" poi a destra dell'assegnazione, questo implica quindi che nonostante lo standard C++ non specifichi se la chiamata di funzione viene valutata per prima o per seconda, il risultato non dipende dall'ordine con cui vengono valutate le due parti.
(PS: normalmente viene valutata prima la chiamata di funzione).
È quindi corretto e stamperà: 13345
(vedere schema)



Esercizi di Teoria per Casa (aa 2014-2015)
02 giugno 2015

(1) Cercate di scrivere una PRE ed una POST sensate per il seguente programma ricorsivo:

```
int F(nodo*L, int k1, int k2) {
if(L)
  if(k2>0)
    return F(L,k1,k2-1);
  else
    return k1+F(L->next,k1,k1);
else
  return k1;
}
```

(2) Considerate il seguente programma. In caso pensiate sia corretto, spiegate cosa calcola e cosa stampa. Se pensate sia sbagliato, spiegate in dettaglio gli errori. In ogni caso, per spiegare le vostre conclusioni, mostrate la relazione tra le variabili e i puntatori in gioco con un disegno. In nessun caso verranno assegnati punti per risposte senza spiegazioni.

```
int* f(int *& p){int b=0,*x=&b; x=p+2; p++; return x-3; }
main() {int b[]={1,2,3,4,5},*q=b+2; *f(q)=*q; cout<<b[0]<<b[1]<<b[2]<<b[3]<<b[4];}
```

(3) Considerate le seguenti conversioni:

```
int x=3, *pi; double y=3.1,*pd=&y;
x=y; pi=reinterpret_cast<int*>(pd);
```

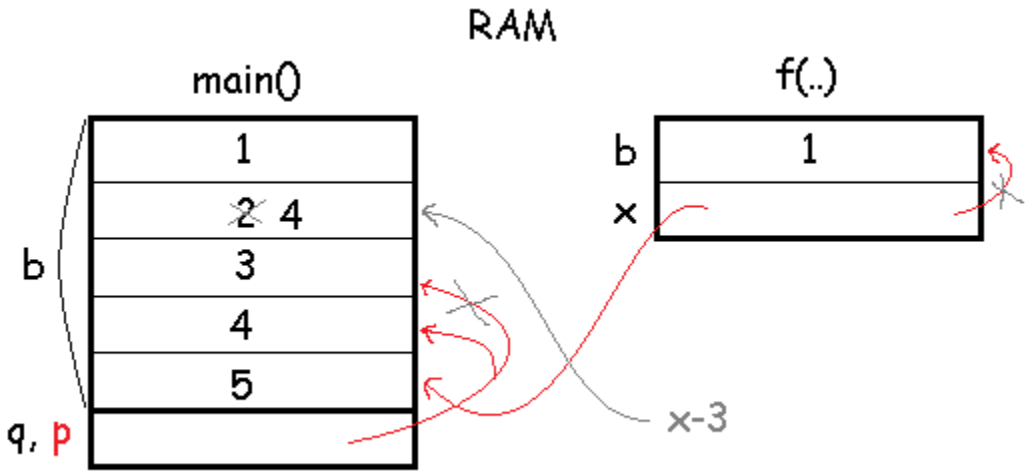
Entrambe le assegnazioni sono accettate dal compilatore? Spiegare la risposta. In caso ce ne fossero di accettate, che valore assumerebbero le variabili coinvolte e anche gli oggetti puntati dopo l'esecuzione delle corrispondenti assegnazioni?

Nota:
oltre che alla pagina seguente le soluzioni ed i testi sono reperibili anche all'indirizzo:
http://pastebin.com/N3R4HWEb

SOLUZIONE Esercizi di Teoria per Casa (aa 2014-2015)
02 giugno 2015
(soluzione by Filè)

(1)
PRE=(lista(L) corretta, k1 e k2 definiti).
POST=(detto n il numero di nodi di lista(L) allora ritorna il valore k=k1*(n+1)).

(2)
(semi)corretto, vale la pena notare che:
tutti i tipi mathcano, e tutti gli accessi sono legali, tramite side-effect vengono spostati i puntatori e avviene una sovrascrittura di una cella dell'array. stampa il contenuto dell'array b[0..4] alla fine di tutto il processo.
tuttavia c'è un'ambiguità:
nel main() l'istruzione *f(q)=*q; è ambigua, dato che il C++ non specifica quale parte viene "valutata" per prima, ovvero se viene prima eseguita l'invocazione (parte sinistra) o prima "pescato" il valore da mettere alla fine nella locazione indicata (parte destra).
normalmente viene eseguita l'invocazione (parte sinistra) per prima, ma lo standard C++ non lo garantisce, delegando tale scelta alla singola implementazione del compilatore.
se viene eseguita prima la parte sinistra, allora stamperà: 14345
altrimenti, se viene eseguita prima la parte destra, allora stamperà: 13345
(lo schema sottostante si riferisce alla situazione in cui viene valutata prima la parte sinistra dell'assegnazione, altrimenti il "2" in b[1] è sostituito con un "3" anziché con un "4").



(3) dette:
a) x=y;
b) pi=reinterpret_cast<int*>(pd);
le due conversioni da analizzare, allora:
a) si vuole assegnare ad una variabile di tipo int un valore di tipo double, il compilatore procede, ma verificandosi una perdita d'informazione (l'int da 4 byte non può "ospitare" tutto il double da 8 byte, pertanto tipicamente la parte decimale viene troncata) dovrebbe emettere un "warning".
b) si vuole assegnare ad un puntatore ad intero l'indirizzo contenuto in un puntatore a double; entrambi i puntatori sono memorizzati in 4 byte per cui non si presenta il problema visto al punto a); il problema è invece sul tipo, e trattandosi di puntatori il C++ non ammette conversione (automatica) alcuna. "reinterpret_cast" forza la conversione: ora pi punterà alla locazione di memoria ove risiede y, tuttavia, accedendo a tale locazione per mezzo di pi, i primi 4 byte del double y verranno interpretati come contenenti un valore int. L'effetto (ad esempio di cout<<*pi) dipende dal tipo di rappresentazione che viene adottato per i valori int e per i valori double (ad esempio: "complemento a 2" e/o "virgola mobile").

Laboratorio 1 (aa 2014-2015)
20 marzo 2015

(1) Dato un array int X[3][4][5][10], nel seguito dovete rappresentare l'R-valore di X con X. Si chiede di:
i) specificare il tipo di X e la dimensione dell'oggetto puntato
ii) specificare il valore e il tipo di (*X)+2 e di *(X+2)
iii) specificare il valore e il tipo di *(X[-8])+8

SOLUZIONE Laboratorio 1 (aa 2014-2015)
20 marzo 2015
(soluzione by Filè)

(1)
i) tipo di X = int (*)[4][5][10]
   dimensione oggetto puntato = 4*5*10*sizeof(int) = 4*5*10*4
ii) tipo di (*X)+2 = tipo di *X = int(*)[5][10]
   valore di (*X)+2 = X+2*(5*10)*4
   tipo di *(X+2) = lo stesso di prima = int(*)[5][10]
   valore di *(X+2) = X[2] = X+2*(4*5*10)*4
iii) tipo di *(X[-8])+8 = int(*)[10]
   valore di *(X[-8])+8 = *(*(X-8))+8 = X-8*(4*5*10)*4+8*(10)*4

Nota:
Una variante di questa soluzione, con qualche commento in più è reperibile al seguente link:
http://pastebin.com/BgcgRfSJ

Laboratorio 2 (aa 2014-2015)
27 marzo 2015

(1) Dire se il seguente programma è corretto o no, spiegando cosa stampa, in caso pensiate sia corretto . Se invece pensate che non sia corretto, spiegate cosa c'è che non va.

```
char & C(char &x, char &y) {y=x; return x;}
main(){char A[] = {'a','b','c'}; C(A[0],A[1]) = A[2]; cout<<A[0]<<A[1]<<A[2]<<endl;}
```

(2) Dire se il seguente programma è corretto o no, spiegando cosa stampa, in caso pensiate sia corretto . Se invece pensate che non sia corretto, spiegate cosa c'è che non va.

```
char * C(char *x, char &y) {x=&y; return x;}
main(){char A[] = {'a','b','c'}; *C(A,A[1]) = *(A+2); cout<<A[0]<<A[1]<<A[2]<<endl;}
```

SOLUZIONE Laboratorio 2 (aa 2014-2015)
27 marzo 2015
(soluzione by Filè)

(1)
La funzione C ritorna un char per riferimento e accetta due char per riferimento come parametri formali. Ritorna x il che è corretto essendo x passato per riferimento dunque "sopravvive" alla de-allocazione delle variabili locali di f.
Quindi C è corretta, nel main vengono passati alla funzione i valori "a" e "b", A[1] diventa alias di A[0] quindi viene ritornato A[0] a questo punto l'array contiene (a, a, c) in A[0] viene assegnato "c" e l'array diventa (c, a, c). Quindi La funzione è corretta, il main pure e stampa "cac".

(2)
La funzione ritorna un puntatore char, come parametri formali chiede un puntatore ed una variabile passata per riferimento (invocazione nel main corretta).
Assegna al puntatore x l'L-valore di y (A[1] in questo caso) che esiste sempre essendo passato per riferimento e ritorna il puntatore. Quindi C è corretta, al suo ritorno viene assegnato il valore di A[2] ("c") in A[1] (il valore ritornato da C) quindi l'array sarà (a, c, c) e verrà stampato "acc".

Nota:
Una variante basata solo su commenti del codice di questa soluzione è reperibile al seguente link:
http://pastebin.com/MVCVDAwY

Laboratorio 3 (aa 2014-2015)
17 Aprile 2015

(1) Si consideri la seguente dichiarazione del tipo mix: struct mix{char c; int x; double z;}
a) che succede se, dopo la dichiarazione di mix, la usiamo nel modo seguente? mix a, b('a',1,1.2);
b) dire che succede se si esegue: mix a; cout<<a;
c) che succede se si esegue mix a; cin>>a; e si inseriscono 'a', 1 e 1.2 da tastiera?

(2) Qual è il valore di X alla fine dell'esecuzione del seguente switch?

```
enum colore{bianco, nero, giallo, rosso, blu} X=bianco;
switch(X) {
case bianco: case nero: X=giallo; X=nero;
case giallo: case rosso: X=bianco; break;
default: X=blu;
}
```

(3) Se abbiamo un programma costituito da 4 parti che sono sviluppate da programmatori diversi.
a) Cos'è necessario per il lavoro di ciascun programmatore?
b) Come si organizzano le 4 parti in modo che le esigenze (a) siano soddisfatte?
c) Che problema pone il fatto che una delle parti definisca una variabile globale che debba essere disponibile anche alle altre parti?

(4) Quali sono le istruzioni del C++ che riguardano le eccezioni e che scopo ha ciascuna di esse?

SOLUZIONE Laboratorio 3 (aa 2014-2015)
17 Aprile 2015
(soluzione NON verificata)

(1)
a) La dichiarazione di “a” va bene, mentre la dichiarazione di “b” è sbagliata perché non c’è il costruttore con i parametri. Come soluzione si può introdurre un costruttore con parametri.
b). Darà un messaggio di errore perché il compilatore non sa come stamparlo essendo un tipo definito dall’utente.
c). Uguale alla b).

(2)
X entra nel case bianco/nero, esegue X=giallo e X=nero -> X diventa nero.
A questo punto salta i test dei casi successivi e, uno alla volta, esegue le varie istruzioni, fino a che non trova un break. Alla fine dunque, X sarà nuovamente bianco.

(3)
a) È necessario che ogni programmatore possa compilare indipendentemente dagli altri la sua parte, e che ciascuna parte sia in grado di usare le “cose” che sono dichiarate e definite nelle altre parti.
b) Ciascuno dei 4 programmatori deve produrre un file header.h contenente le dichiarazioni ed un file .cpp che conterrà le definizioni. Il file .h dovrà contenere i prototipi delle funzioni ed tipi dichiarati nella parte di ciascun programmatore.
L’idea è che ciascun programmatore includa il proprio header e quelli dei colleghi nel proprio file .cpp, così dovranno fare anche gli altri.
c) La variabile globale deve essere dichiarata solamente in uno degli header.
Negli header degli altri dovrà dunque esserci la dichiarazione della variabile globale preceduta dalla keyword extern.
(vedi libro di testo, capitolo 9)

(4)
Try, catch e throw.
Try è sempre seguita da un blocco, alla fine del quale c’è l’istruzione di catch che serve a prendere le eccezioni sollevate all’interno del try.
L’istruzione di throw serve per sollevare l’eccezione ed è accompagnata da un valore.
Se vi fossero diversi catch a disposizione di una throw verrà eseguito l’ultimo tra quelli sollevati.
(vedi libro di testo, capitolo 9)