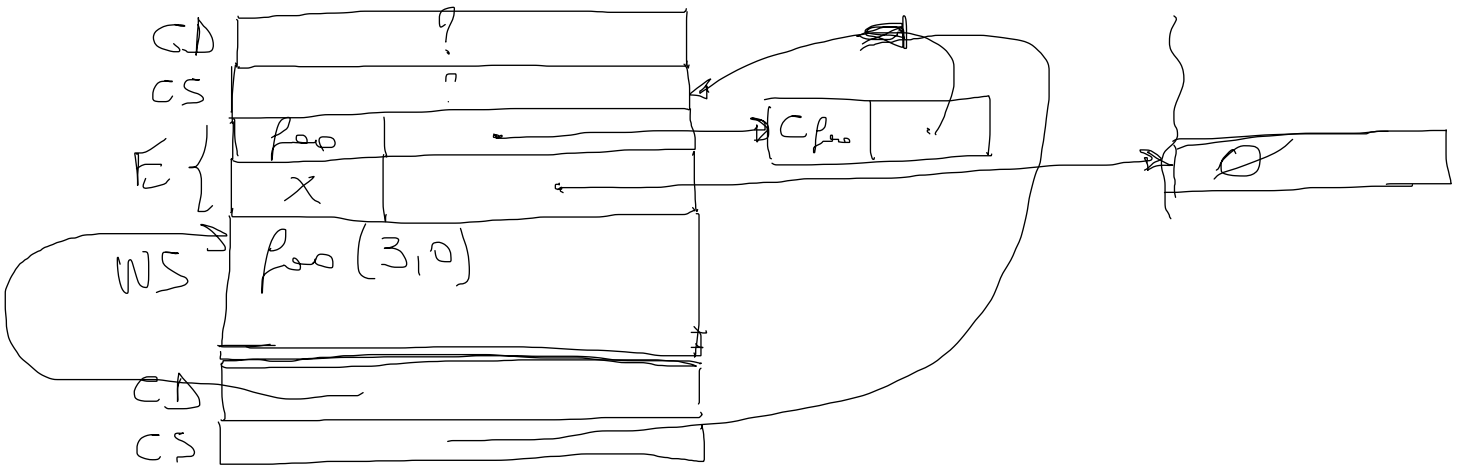
# AltriEsercizi4 (lunedí 11/4/2016) Esercizio1

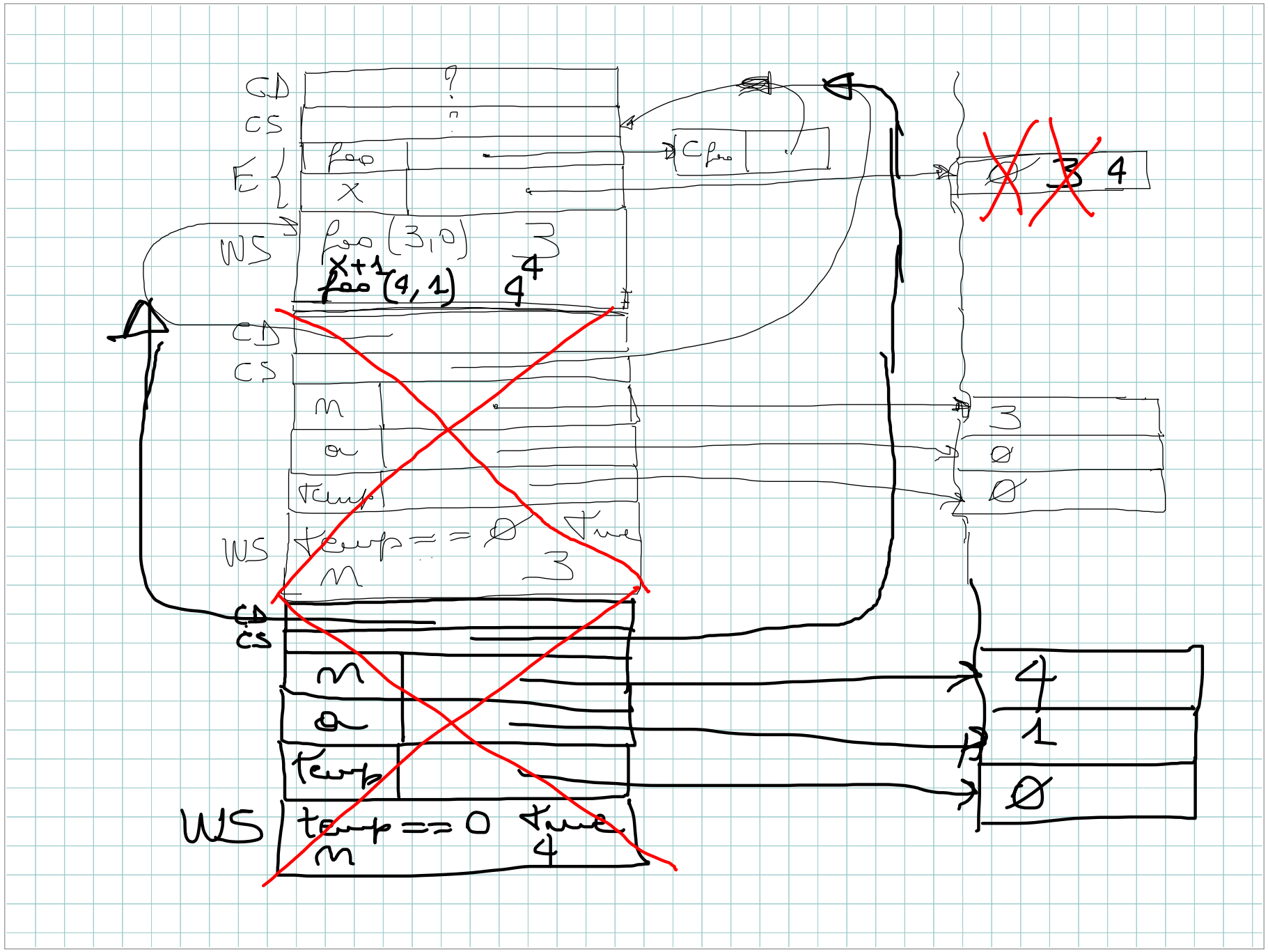
**Esercizio**

*Si mostri la sequenza di AR generata dalla valutazione del seguente frammento di programma di un linguaggio a blocchi, scope statico, trasmissione per valore e, struttura C-like di comandi ed espressioni:*

```
int foo(int n, int a){
  int temp = 0;
  if (temp == 0)return n;
  else return n + 1;
}
...
int x;
x = foo(3, 0);
x = foo(x + 1, 1);
```

*In particolare si mostri come cambia il valore associato a* `x` *nei vari AR.*




CD
CS
?
foo
x
foo (3,0)
3
x+1
foo (4,1)
4
WS
CD
CS
m
a
temp
temp == 0
true
m
3
WS
CD
CS
m
a
temp
temp == 0
true
m
4
foo
3
4
3
0
0
4
1
0

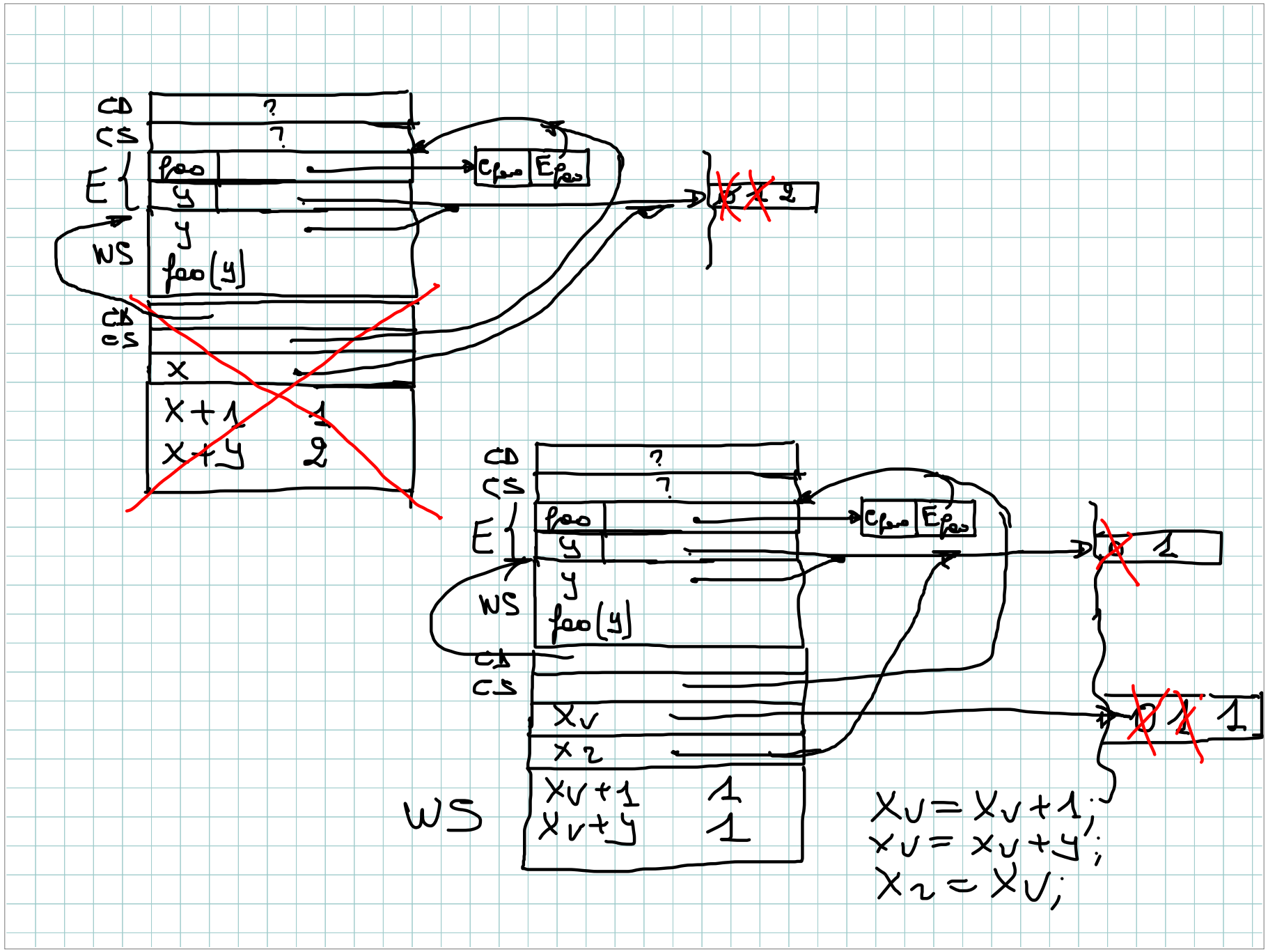
# AltriEsercizi4 (lunedí 11/4/2016) Esercizio2

**Esercizio**

*Si mostri la sequenza di AR generata dalla valutazione del seguente frammento di programma di un linguaggio a blocchi, scope statico, trasmissione per xxxxx e, struttura C-like di comandi ed espressioni:*

```
int y = 0;
void foo(int xxxxx x){
  x = x + 1;
  x = x + y;
}
y = 0;
foo(y);
```

*In particolare si mostri come cambia il valore associato a* `y` *nei vari AR, quando xxxxx è trasmissione per:*
*(1) reference.*
*(2) value-result*


CD
CS
E
foo
y
WS
foo(y)
x
x+1
x+y
1
2
Cfoo
Efoo
xv
x2
xv+1
xv+y
xv=xv+1;
xv=xv+y;
x2=xv;

# AltriEsercizi4 (lunedí 11/4/2016) Esercizio3

**Esercizio**

*(a) Si mostri la struttura degli AR generati dalla valutazione del seguente frammento di programma C:*

```
int y = 0;
void foo(int * x){
  *x = *x + 1;
  *x = *x + y;
}
y = 0;
foo(&y);
```

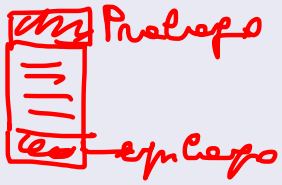
*(b) Si confronti tale struttura con quella ottenuta in esercizio 2.1.*

# AltriEsercizi4 (lunedí 11/4/2016) Esercizio4

## Esercizio

*Si utilizzi la* copy rule *per mostrare come dovrebbe essere rifrasato il seguente frammento di programma di un linguaggio a blocchi, scope statico, trasmissione per valore e per nome e, struttura C-like di comandi ed espressioni:*

```
{...
    int summ(name int exp; name int i; value int start; value int stop){
      int acc = 0;
      for (i = start, i <= stop, i++) acc = acc + exp;
      return acc;
    }
  ...
      {int x = ...
       ...
       int y = summ((x − 1) ∗ (x − 1), x, 1, 10);
       ...
      }
      {int z = ...
       ...
       int y = summ(z ∗ z, z, 3, 12);
       ...
      }
```



# AltriEsercizi4 (lunedí 11/4/2016) Esercizio4 - soluzione

**Esercizio**

*Si utilizzi la* copy rule *per mostrare come dovrebbe essere rifrasato il seguente frammento di programma di un linguaggio a blocchi, scope statico, trasmissione per valore e per nome e, struttura C-like di comandi ed espressioni:*

```
{...
    int summ(name int exp; name int i; value int start; value int stop){
      int acc = 0;
      for (i = start, i <= stop, i++) acc = acc + exp;
      return acc;
    }
 ...
      {int x = ...
       ...
       int y = summ((x − 1) ∗ (x − 1), x, 1, 10);
       ...
      }
      ...
```

Soluzione(parziale)

```
 ...
      {int x = ...
       ...
       int y;
       {/∗ int y = summ((x − 1) ∗ (x − 1), x, 1, 10); ∗/
        int start = 1;
        int stop = 10;
        int acc = 0;
        for (x = start, x <= stop, x++) acc = acc + (x − 1) ∗ (x − 1);
        y = acc;
       }
       ...
      }
```

# AltriEsercizi4 (lunedí 11/4/2016) Esercizio5

**Esercizio**

*Si mostri la sequenza di AR generata dalla valutazione del seguente frammento di programma di un linguaggio a blocchi, scope statico, trasmissione per valore e per funzione, struttura C-like di comandi ed espressioni:*

```
{
    int z = 10;
    int y = 3;
    int g(int x){;
      if (x >= y) y = y + z;
      else y = y − x;
      return y − 5;
    }
    void h(int x, int y){;
      if (x >= y) z = g(x);
      else y = g(y − x) + 3;
      print (x, y, z);
    }
    h(g(y), z);
    print (y);
}
```

*In particolare si mostri quali valori sono stampati e come sono ottenuti.*

# AltriEsercizi3 (lunedí 4/4/2016) Esercizio4

**Esercizio**

*Si consideri la versione del fattoriale memoized visto a lezione (slide 22 del 23/3/2016).*

*(a) Si discuta il ruolo delle dichiarazione static nelle strutture dati introdotte nella definizione di memoizedFact.*

*(b) Si dica come cambia a funzione calcolata da memoizedFact se il modificatore static rimosso.*

- Ci aspettiamo che la memoria relativa sia allocata dinamicamente ad ogni nuova creazione di un Activation Record.
- Di conseguenza l'esecuzione condurrà a... (prendi il listing allegato e fai modifiche e test significativi)

# AltriEsercizi2 (lunedì 4/4/2016) Esercizio5

**Esercizio**

*Si consideri il listing* `DerivaPack` *allegato contenente una versione strutturata in moduli C del programma Deriva. Si estenda il programma con una versione tail-recursive dell'operazione* `size` *che calcola il numero di alberi di un treeList. In particolare, si:*

*a) Scriva ed aggiunga size nell'opportuno modulo*

*b) Ri-compili e ri-esegua il programma Deriva*

(a)

- Ci aspettiamo di modificare il modulo ...
- Scarichiamo il Linsting e procediamo

(b)

- Aggiungiamo, nel main, codice opportuno per un test della nuova operazione

# AltriEsercizi2 (lunedì 4/4/2016) Esercizio7

**Esercizio**

*Si discuta un programma per la definizione di tabelle hash a liste di collisione e con funzione hash a divisione.*

- L'esercizio6 precedente mostra che l'implementazione del fattoriale con memoization richiede tabelle di memorizzazione di modeste dimensioni e che l'aritmetica delle macchine attuali non operano con fattoriale oltre un piccolo naturale (prossimo a 20)
- Nondimeno, l'impiego di memoization può richiedere l'uso di tabelle hash per rappresentare la mappa finita delle applicazioni di funzione correntemente utilizzate dalla computazione. Limitandoci a funzioni monodiche, l'argomento una chiave, e l'immagine della funzione il valore associato alla chiave.
- **Tabelle a liste di collisione**: Hanno chiavi condivise da più argomenti (distr. unifor.) ed associato ad ogni chiave una lista di coppie $(a_n, v_n)...(a_1, v_1)$, dove $a_n$ è l'n-esimo argomento con tale chiave utilizzato, nella computazione del programma, come argomento della funzione, e $v_n$ è l'immagine calcolata.
- **Funzione hash a divisione**: `chiave(a) = P mod a`, dove P è un intero (primo non prossimo a potenza di 2).
- Implementare in C per il prossimo lunedì 11/4/2016, con operazioni:
  `add, remove, find`.