

PROCEDURE

Una *procedura* permette di

- dare un nome a una istruzione
- rendendola *parametrica*
- non denota un valore, quindi non c'è tipo di ritorno → **void**

```
void p(int x) {  
    float y = x * sin(0.75);  
}
```

1

PROCEDURE COME SERVITORI

Una *procedura* è un *componente software* che cattura l'idea di “macro-istruzione”

- molti possibili parametri, che **possono anche essere modificati** mentre nelle funzioni normalmente **non devono** essere modificati
- nessun “valore di uscita” esplicito

Come una funzione, una procedura è un servitore

- *passivo*
 - che serve *un cliente per volta*
 - che può trasformarsi in cliente *invocando se stessa o altre procedure*
- In C, una procedura ha la stessa struttura di una funzione, salvo il **tipo di ritorno** che è **void**

2

PROCEDURE

L'istruzione *return* provoca solo la restituzione del controllo al cliente e non è seguita da una espressione da restituire -> *non è necessaria* se la procedura termina "spontaneamente" a fine blocco

Nel caso di una procedura, non esistendo valore di ritorno, cliente e servitore comunicano solo:

- mediante ***parametri***
- mediante ***aree dati globali***

➡ Occorre il ***passaggio per riferimento*** per fare cambiamenti permanenti ai dati del cliente

3

PASSAGGIO DEI PARAMETRI

In generale, un parametro può essere trasferito dal cliente al servitore:

- **per valore o copia (*by value*)**
si trasferisce il valore del parametro attuale
- **per riferimento (*by reference*)**
si trasferisce un riferimento al parametro attuale

4

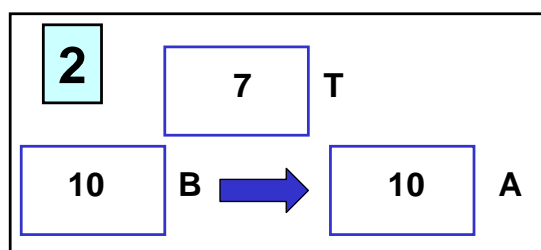
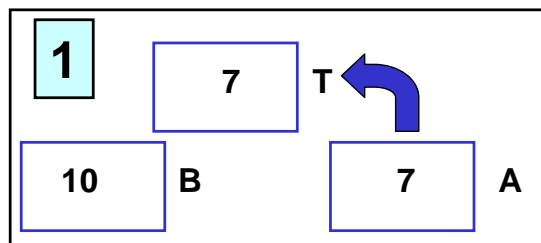
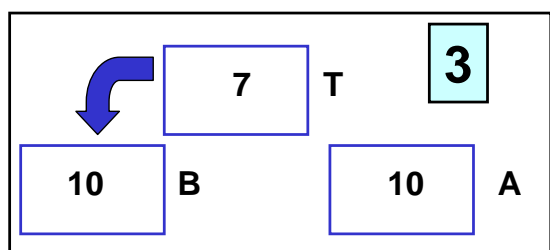
ESEMPIO

Perché il passaggio per valore non basta?

Problema: scrivere una procedura che *scambi i valori di due variabili intere*

Specifica:

Dette A e B le due variabili, ci si può appoggiare a una *variabile ausiliaria T*, e svolgere lo scambio in *tre fasi*



ESEMPIO

Supponendo di utilizzare, senza preoccuparsi, il passaggio per valore usato finora, la codifica potrebbe essere espressa come segue:

```
void scambia(int a, int b) {  
    int t;  
    t = a;    a = b;    b = t;  
    return; /* può essere omessa */  
}
```

ESEMPIO

Il cliente invocherebbe quindi la procedura così:

```
main(){
  int y = 5, x = 33;
  scambia(x, y);
  /* ora dovrebbe essere
    x=5, y=33 ...
    MA NON È VERO
  */
}
```

Perché non funziona?

7

ESEMPIO

- La procedura ha *effettivamente scambiato* i valori di A e B al suo interno
- ma questa modifica non si è propagata al cliente, perché sono state scambiate *le copie locali alla procedura, non gli originali*
- al termine della procedura, le sue variabili locali sono state distrutte → nulla è rimasto del lavoro svolto dalla procedura

X

33

Y

5

~~A

5

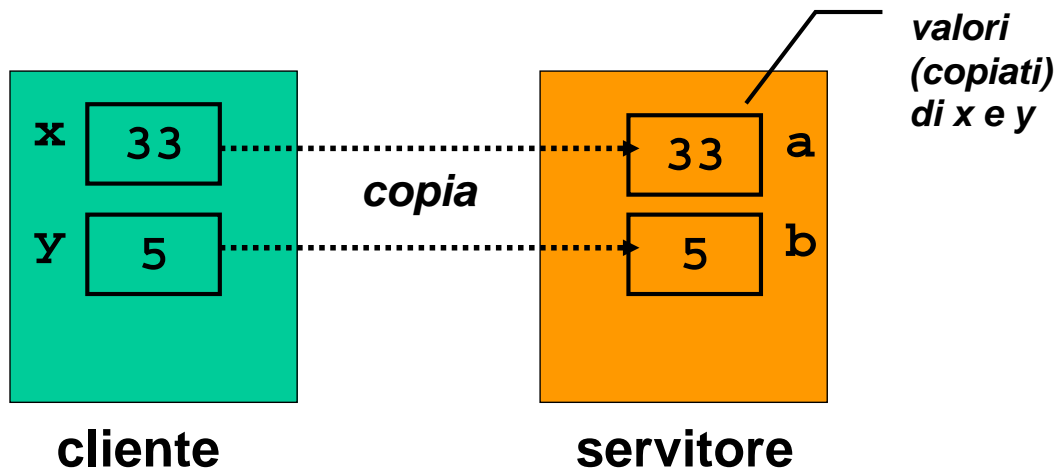
B

33

8

PASSAGGIO PER VALORE

Ogni azione fatta su *a* e *b* è strettamente locale al servitore. Quindi *a* e *b* vengono scambiati ma quando il servitore termina, tutto scompare



9

PASSAGGIO DEI PARAMETRI IN C

Il C adotta **sempre il passaggio per valore**

- le variabili del cliente e del servitore sono ***disaccoppiate***
- ma *non consente di scrivere componenti software il cui scopo sia diverso dal calcolo di una espressione*
- per superare questo limite occorre il ***passaggio per riferimento (by reference)***

10

PASSAGGIO PER RIFERIMENTO

Il passaggio per riferimento (*by reference*)

- NON trasferisce ***una copia del valore*** del parametro attuale
- ***ma un riferimento al parametro***, in modo da dare al servitore accesso diretto al parametro in possesso del cliente
- il servitore, quindi, ***accede direttamente*** al dato del cliente e ***può modificarlo***

11

PASSAGGIO DEI PARAMETRI IN C

Il linguaggio C ***NON*** supporta *direttamente* il ***passaggio per riferimento***

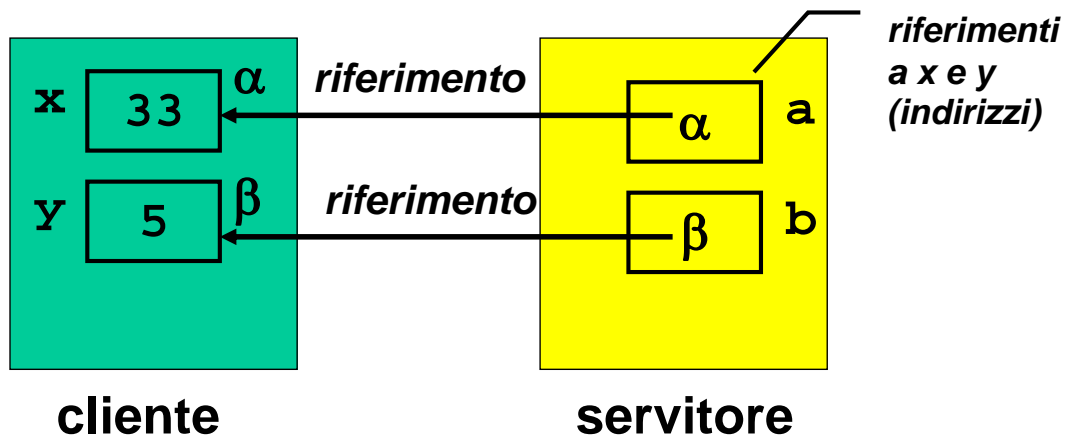
- è una grave mancanza
- viene fornito indirettamente solo per alcuni tipi di dato
- occorre quindi ***costruirlo quando serve***

12

PASSAGGIO PER RIFERIMENTO

Si trasferisce un riferimento ai parametri attuali (cioè i loro indirizzi)

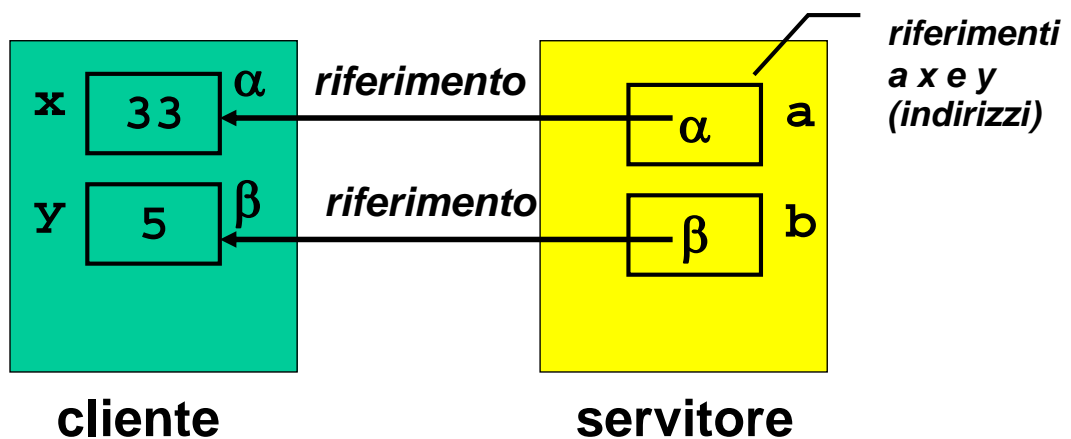
Ogni azione fatta su **a** e **b** in realtà è fatta su **x** e **y** nell'environment del cliente



13

PASSAGGIO PER RIFERIMENTO

Quindi, scambiando **a** e **b** in realtà si scambiano **x** e **y**



14

REALIZZARE IL PASSAGGIO PER RIFERIMENTO IN C

Il C *non* fornisce *direttamente* un modo per attivare il passaggio per riferimento -> a volte occorre *costruirselo*

È possibile costruirlo? Come?

- Poiché passare un parametro per riferimento comporta la capacità di manipolare **indirizzi di variabili...**
- ... gestire il passaggio per riferimento implica la capacità di accedere, *direttamente o indirettamente*, agli indirizzi delle variabili

15

REALIZZARE IL PASSAGGIO PER RIFERIMENTO IN C

In particolare occorre essere capaci di:

- **ricavare l'indirizzo** di una variabile
- **dereferenziare un indirizzo** di variabile, ossia "recuperare" il valore dato l'indirizzo della variabile
- Nei linguaggi che offrono direttamente il passaggio per riferimento, *questi passi sono effettuati* in modo trasparente all'utente
- In C il **programmatore deve conoscere gli indirizzi** delle variabili e quindi accedere alla macchina sottostante

16

INDIRIZZAMENTO E DEREFERENCING

Il C offre a tale scopo *due operatori*, che consentono di:

- *ricavare l'indirizzo* di una variabile
operatore estrazione di indirizzo **&**
- *dereferenziare un indirizzo* di variabile
operatore di dereferenziamento *****

17

INDIRIZZAMENTO E DEREFERENCING

Se **x** è una variabile,
&x denota l'*indirizzo in memoria* di tale
variabile:

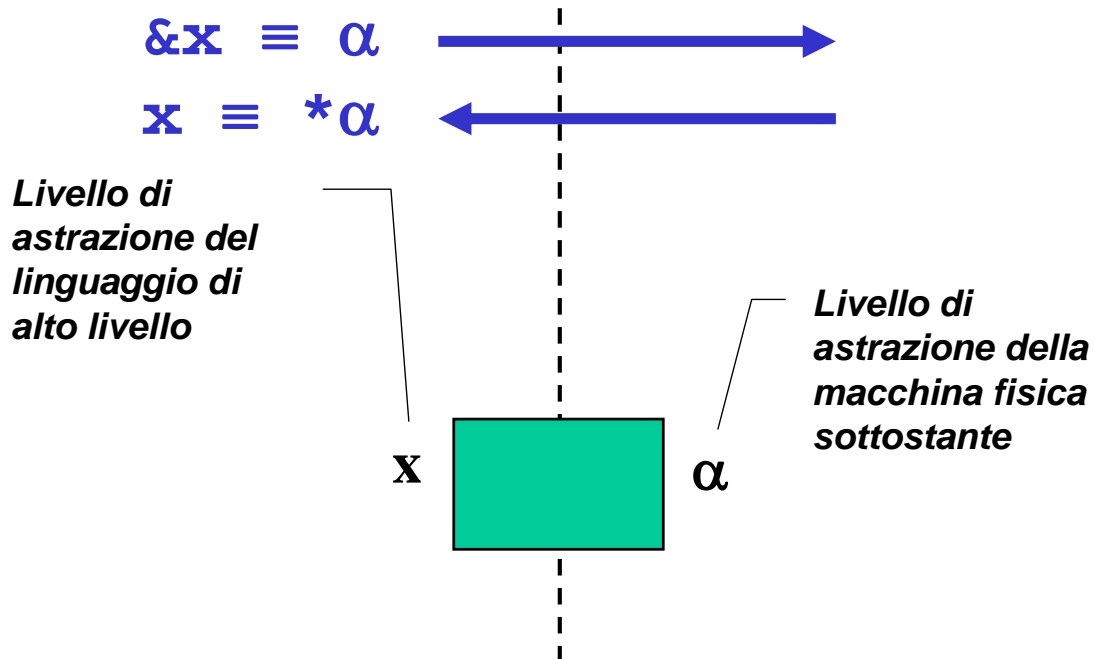
$$\mathbf{\&x} \equiv \alpha$$

Se α è l'indirizzo di una variabile,
*** α** denota *tale variabile*:

$$\mathbf{x} \equiv \mathbf{*}\alpha$$

18

INDIRIZZAMENTO E DEREFERENCING



19

PUNTATORI

Un **puntatore** è il costrutto linguistico introdotto dal C (e da altri linguaggi) come *forma di accesso alla macchina sottostante* e in particolare agli **indirizzi di variabili**

- Un **tipo puntatore a T** è un tipo che denota l'indirizzo di memoria di una variabile di tipo T
- Un **puntatore a T** è una variabile di "tipo puntatore a T" che può contenere l'indirizzo di una variabile di tipo T

20

PUNTATORI

Definizione di una variabile puntatore:

```
<tipo> * <nomevariabile> ;
```

Esempi:

```
int *p;  
int* p;  
int * p;
```

Queste tre forme sono equivalenti, e definiscono p come “puntatore a intero”

21

PASSAGGIO PER RIFERIMENTO IN C

- il cliente deve passare esplicitamente gli indirizzi
- il servitore deve prevedere esplicitamente dei puntatori come parametri formali

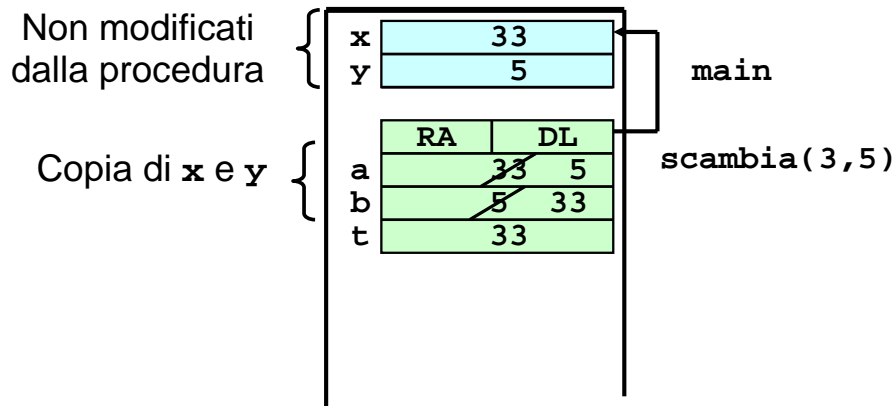
```
void scambia(int* a, int* b) {  
    int t;  
    t = *a; *a = *b; *b = t;  
}
```

```
main(){  
    int y=5, x=33;  
    scambia(&x, &y);  
}
```

22

ESEMPIO: RECORD DI ATTIVAZIONE

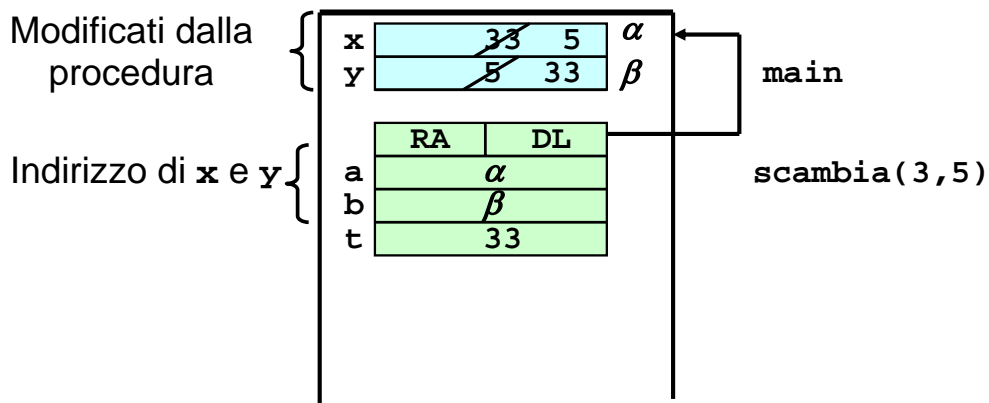
Caso del *passaggio per valore*:



23

ESEMPIO: RECORD DI ATTIVAZIONE

Caso del *passaggio per riferimento*:



24

OSSERVAZIONE

- Quando un puntatore è usato per realizzare il passaggio per riferimento, *la funzione **non dovrebbe mai alterare il valore del puntatore***
- Quindi, se **a** e **b** sono due puntatori:

***a = *b** **SI**

~~**a = b**~~ **NO**

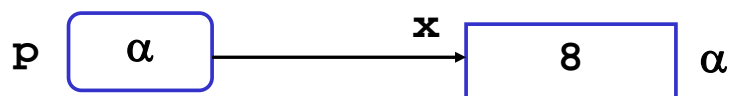
- In generale una funzione PUÒ modificare un puntatore, ma *non è opportuno che lo faccia se esso realizza un passaggio per riferimento*

25

PUNTATORI

- Un **puntatore** è una variabile *destinata a contenere l'indirizzo* di un'altra variabile
- Vincolo di tipo: un puntatore a T può contenere solo l'indirizzo di variabili di tipo T

Esempio:



```
int x = 8;  
int* p;  
p = &x;
```

Da questo momento, ***p** e **x** sono **due modi alternativi per denotare la stessa variabile**

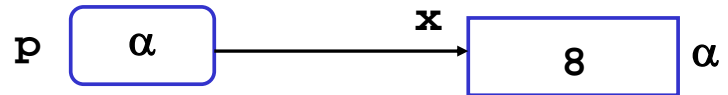
26

PUNTATORI

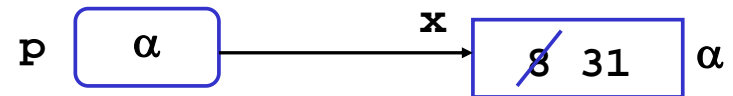
```
int x = 8;
```



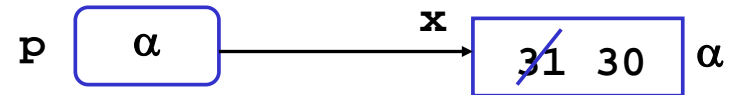
```
int* p = &x;
```



```
*p = 31;
```



```
x--;
```



27

PUNTATORI

Un puntatore non è legato per sempre alla stessa variabile; può essere modificato

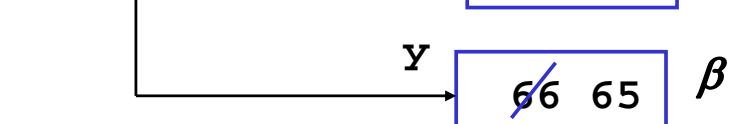
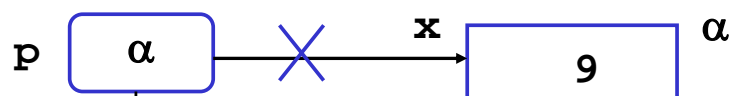
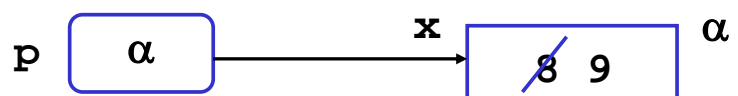
```
int x = 8, y = 66;
```

```
int *p = &x;
```

```
(*p)++;
```

```
p = &y;
```

```
(*p)--;
```



Le parentesi sono necessarie? Che cosa identifica la scrittura *p-- ?

28

PUNTATORI

Un puntatore a T può contenere solo l'indirizzo di variabili di tipo T: **puntatori a tipi diversi sono incompatibili tra loro**

Esempio:

```
int x=8, *p; float *q;  
p = &x;      /* OK */  
q = p;      /* NO! */
```

MOTIVO: il tipo del puntatore serve per dedurre il tipo dell'oggetto puntato, che è una **informazione indispensabile per effettuare il dereferencing**

29

PUNTATORI

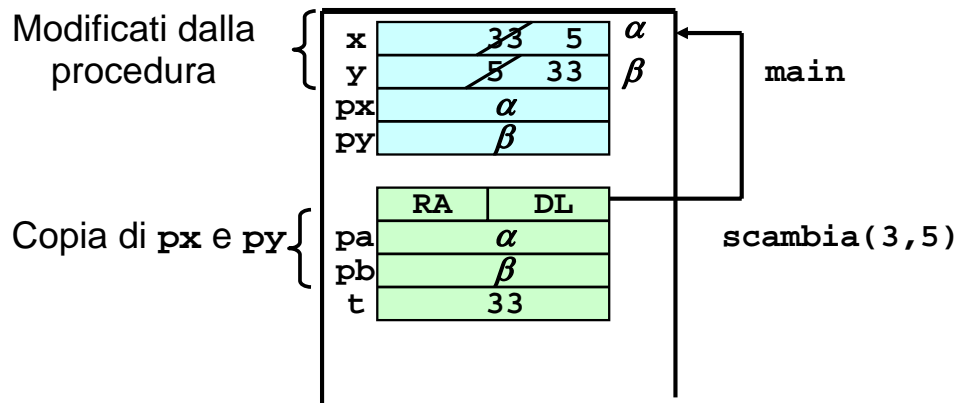
```
void scambia(int* pa, int* pb) {  
    int t;  
    t = *pa; *pa = *pb; *pb = t;  
}  
  
main(){  
    int y = 5, x = 33;  
    int *py = &y, *px = &x;  
    scambia(px, py);  
}
```

Variazione dall'esempio precedente: i puntatori sono memorizzati in `px` e `py` prima di passarli alla procedura

30

ESEMPIO: RECORD DI ATTIVAZIONE

Il record di attivazione si modifica come segue



31

COMUNICAZIONE TRAMITE ENVIRONMENT GLOBALE

Una procedura può anche comunicare con il cliente **mediante aree dati globali**: ad esempio, **variabili globali**

Le **variabili globali** in C:

- sono allocate **nell'area dati globale** (fuori da ogni funzione)
- esistono **prima** della chiamata del **main**
- sono **inizializzate automaticamente a 0** salvo diversa indicazione
- possono essere **nascoste** in una funzione da una variabile locale omonima
- sono visibili, previa dichiarazione **extern**, in tutti i file dell'applicazione

32

ESEMPIO

Esempio: Divisione intera x/y con calcolo di quoziente e resto. Occorre calcolare *due* valori che supponiamo di mettere in due variabili globali

```
int quoziente, int resto;
```

variabili globali quoziente e resto visibili in tutti i blocchi

```
void dividi(int x, int y) {  
    resto = x % y; quoziente = x/y;  
}
```

```
main(){  
    dividi(33, 6);  
    printf("%d%d", quoziente, resto);  
}
```

Il risultato è disponibile per il cliente nelle variabili globali quoziente e resto

33

SOLUZIONE ALTERNATIVA

Esempio: Con il passaggio dei parametri per indirizzo avremmo il seguente codice

```
void dividi(int x, int y, int* quoziente,  
            int* resto) {  
    *resto = x%y;    *quoziente = x/y;  
}
```

```
main(){  
    int k = 33, h = 6, quoz, rest;  
    int *pq = &quoz, *pr = &rest;  
    dividi(33, 6, pq, pr);  
    printf("%d%d", quoz, rest);  
}
```

34