Fondamenti di Informatica e Laboratorio T-AB
Ingengeria dell'Automazione
a.a. 2008/2009

Lab 02
Tipi semplici in C

Obiettivo dell’esercitazione

- Acquistare familiarità con i tipi di dato semplici supportati dal linguaggio C
- Comprenderne alcuni limiti nella rappresentazione dell’informazione
  - Dimensione in byte dei tipi semplici e limiti di rappresentazione
  - Problemi di overflow, underflow, troncamento e “division by zero”
  - Espressioni su interi, razionali, e casting esplicito

Quanti bit sono usati per un tipo?

- In C il numero di bit utilizzati per ogni tipo dipende dal compilatore
- Uniche regole:
  - short int: almeno 16 bit (2 byte)
  - int: a discrezione del compilatore, ma vale sempre:

    sizeof(int) >= sizeof (short int)
  - long int: almeno 32 bit (4 byte), e vale sempre

    sizeof(long int) >= sizeof(int)

Quanti bit sono usati per un tipo?

- float: nessun limite minimo, ma tipicamente almeno 32 bit (4 byte)
- double: nessun limite minimo, ma tipicamente almeno 64 bit (8 byte)
- long double: ???

Quanti bit sono usati per un tipo?

Come posso conoscere le dimensioni di un tipo?

1. Gli header “limits.h” e “float.h” specificano le costanti tipiche di un compilatore (vedremo nelle prossime lezioni cosa sono gli *header files*)

2. Posso usare l’operatore sizeof

sizeof è un operatore speciale del linguaggio C, che applicato ad un tipo restituisce il numero di **bytes** usati per memorizzare quel tipo

Esercizio 1

```
#include <stdio.h>
int main(void)
{
   int dim1, dim2, dim3;
   int dim4, dim5, dim6;

   dim1 = sizeof(short int);
   dim2 = sizeof(int);
   dim3 = sizeof(long int);
   dim4 = sizeof(float);
   dim5 = sizeof(double);
   dim6 = sizeof(long double);
   return (0);
}
```

1. Copiare, compilare ed eseguire il seguente programma
2. Utilizzando il debug e le finestre di “watch”/”locals”, rispondere alle seguenti domande:
   a) Quanto vale dim2 *prima* e *dopo*l’esecuzione dell’assegnamento?
   b) Quanti **bit** sono utilizzati per rappresentare un intero?
   c) Quanti **bit** sono utilizzati per rappresentare un float?

Quanti numeri interi posso rappresentare con una variabile di tipo X?

Supponiamo che uno short int sia codificato con 16 bit (2 byte)…

… 16 bit → 2^{16} →ho a disposizione 65536 simboli, ma…

… dobbiamo decidere anche se l'intero è *signed* o *unsigned…*

1. Caso short int (signed short int): -32768 … 32767
2. Caso unsigned short int: 0 … 65535

Esercizio 2

```
#include <stdio.h>
int main(void)
{
   short int i;
   short int k;

   k = 10000;
   i = 30000 + k;

   return (0);
}
```

1. Copiare, compilare ed eseguire il seguente programma

2. Utilizzando il debug e le finestre di “watch”/”locals”, rispondere alle seguenti domande:
   a) Quanto valgono *i* e *k* prima degli assegnamenti?
   b) Secondo voi, quanto dovrebbe valere *i* dopo l’assegnamento?
   c) Quanto vale effettivamente *i* dopo l’assegnamento? Perchè?

3. Modificate il programma, specificando i e k come variabili *unsigned*… cosa cambia? Il comportamento del programma ora è corretto? Perchè?

E’ sempre possibile rappresentare un qualunque numero reale?

Anche la rappresentazione dei numeri reali soffre di alcuni limiti:

1. Indipendentemente da quanti bit uso per rappresentare un numero reale, tali bit devono essere sempre in numero *finito*…
   … se il numero di bit è finito, da qualche parte dovrò approssimare qualcosina…

2. La trasformazione della rappresentazione di un numero reale da una base ad un’altra non è sempre indolore…
   …può succedere che, dato un numero reale con un numero di cifre decimali finito in base 10…
   … durante la trasformazione di base possa diventare un numero con con la parte dopo la virgola addirittura PERIODICA! Quindi, ulteriore approssimazione…

Esercizio 3

```
#include <stdio.h>
int main(void)
{
   float k;

   k = 5.6F;

   k = k - 5.59F;

   return (0);
}
```

1. Copiare, compilare ed eseguire il seguente programma

2. Utilizzando il debug e le finestre di “watch”/”locals”, rispondere alle seguenti domande:
   a) Quanto vale *k* prima del primo assegnamento?
   b) Quanto vale *k* dopo il primo assegnamento? Quant’e’ l’errore di approssimazione?
   c) Quanto dovrebbe valere, e quanto vale effettivamente *k* dopo il secondo assegnamento? Perchè?

3. Modificate il programma, specificando *k* come variabile double… cosa cambia? Quanto vale l’errore di approssimazione?

Esercizio 4

Si utilizzi il seguente schema di programma C:

```
#include<stdio.h>
main() {// dichiarazioni
[<qual>] [<quant>] <tipo> <var> , <var> , ... ;
// ...
// istruzioni
// ...
}
```

Si definiscano le seguenti variabili:

- una variabile A per valori interi con la minima occupazione di memoria possibile, che dovrà avere nel proprio dominio i valori da -50000 a 30000;
- una variabile B per valori interi con la minima occupazione di memoria possibile, che dovrà avere nel proprio dominio i valori da 0 a 215 (ma non valori negativi);
- una variabile C di tipo carattere;
- una variabile D di tipo double

Espressioni eterogenee

Cosa succede se in una espressione uso tipi diversi?

… laddove possibile, una espressione eterogenea viene risolta applicando la *promotion* dei tipi, e considerando *l'overloading* degli operatori…

Cosa succede se assegno ad un tipo *inferiore* un valore rappresentato tramite un tipo *superiore*?

Come si effettua la conversione esplicita da un tipo ad un altro?

Esercizio 5

```
#include <stdio.h>
int main(void)
{
   int i, k;
   float j;

   i = 20;
   k = i % 3;
   i = i / 3;

   k = i / 4.0F;
   j = i / 4.0F;

   return (0);
}
```

1. Copiare, compilare ed eseguire il seguente programma…

2. Utilizzando il debug e le finestre di “watch”/”locals”, rispondere alle seguenti domande:
   a) Quanto valgono *i* e *k* dopo il primo blocco di assegnamenti?
   b) Quanto valgono k e j dopo il secondo blocco di assegnamenti?
   c) Se k e j al termine del programma hanno valori diversi, perchè?

3. In fase di compilazione il programma potrebbe aver generato dei *warnings*…
   a) Leggete i warning e spiegate il loro significato
   b) Correggete il codice al fine di far scomparire i warning (compilate sempre con “rebuild all”)

Esercizio 6

Si assegnino valori iniziali alle variabili nel modo seguente:
A <- 0
B <- 10^4
C <- 77
D <- 10.0
(in linguaggio C, per esprimere il valore 104, si può usare la notazione 1E4)

Si eseguano le seguenti operazioni:
- Si divida B per C (risultato in B)
- Si sottragga D a C (risultato in C)
- Si assegni a C la metà del valore di B
- Si sommi D ad A (risultato in A)
- Si controlli l'esecuzione del programma con in debugger, per
- seguire l'evoluzione del valore delle variabili.
- Che valori visualizza il debugger al termine dell'esecuzione?