Fondamenti di Informatica T-1
Modulo 2

Warning vs Errors (1)

- Il risultato del processo di compilazione è:
  - se il programma è **sintatticamente corretto** → un file oggetto, pronto per essere sottoposto alla fase di *linking*
  - altrimenti → la notifica di una serie di errori
- In entrambi i casi, il compilatore può comunque segnalare dei warning
  - *Potenziali* sorgenti di errori a run-time o di comportamenti non voluti

Warning vs Errors (2)

- Ci sono *warning* più o meno gravi
  - Alcuni non rappresentano un vero problema…
  - …spesso, invece, possono essere una vera fonte di errori a tempo di esecuzione
    - Nota che gli errori che si verificano runtime sono i più difficili da prevedere/gestire
  - Il livello di gravità di un *warning* può essere valutato solo dal programmatore, dipendentemente dal contesto in cui si verifica
- Quindi È FONDAMENTALE CONSIDERARE CON ATTENZIONE TUTTI I WARNING EVENTUALMENTE GENERATI DAL COMPILATORE

Warning vs Errors (3)

Un semplice programma:

```
#include <stdio.h>

int main(void)
{
   int IVAperc = 0.2;
   float prezzo = 11200;
   printf("IVA %f", IVAperc * prezzo);
   return (0);
}
```

Warning vs Errors (4)

- **`float IVAperc = 0.2;`**
  - *warning C4305: 'initializing' : truncation from 'double' to 'float'*
  - Non c'è in realtà alcun troncamento (0.2 non richiede doppia precisione) ma a default le costanti numeriche reali vengono tradotte in double
  - Conviene seguire la sintassi corretta:

    `float f = 0.2f;`
- Supponiamo ora di modificare la definizione in

  `int IVAperc = 0.2;`
  - *warning C4244: 'initializing' : truncation from 'double' to 'int', possible loss of data*
  - 0.2 viene in effetti troncato a 0
  - Nell'operazione **`IVAperc * prezzo`**, **`IVAperc`** viene promosso a float, ma ormai ***la perdita di informazione è avvenuta***
  - ***Il risultato è sempre e comunque 0 (GRAVE)!***

Tipi semplici in C - obiettivo

- 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
(tipi semplici)

```
#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
(tipi semplici)

```
#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
(tipi semplici)

```
#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?