Fondamenti di Informatica T-1
Modulo 2

Obiettivi di questa lezione

1. Alcune considerazioni sulla differenza tra “warnings” ed “errors”

2. Tipi semplici
3. Input e output in linguaggio C
4. Espressioni

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?

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 1
(espressioni)

```
#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")

Operatori: priorità ed associatività

Priorità	Operatore	Simbolo	Associatività
1 (max)	chiamate a funzione selezioni	`()` `[]` `->` `.`	a sinistra

2	operatori unari:
	op. negazione
	op. aritmetici unari
	op. incr. / decr.
	op. indir. e deref.
	op. sizeof
		
		`!` `~`
		`+` `-`
		`++` `--`
		`&` `*`
		`sizeof`
			a destra

3	op. moltiplicativi	`*` `/` `%`	a sinistra
4	op. additivi	`+` `-`	a sinistra

Operatori: priorità ed associatività

Priorità	Operatore	Simbolo	Associatività
5	op. di shift	`>>` `<<`	a sinistra
6	op. relazionali	`<` `<=` `>` `>=`	a sinistra
7	op. uguaglianza	`==` `!=`	a sinistra
8	op. di AND bit a bit	`&`	a sinistra
9	op. di XOR bit a bit	`^`	a sinistra
10	op. di OR bit a bit	`|`	a sinistra
11	op. di AND logico	`&&`	a sinistra
12	op. di OR logico	`||`	a sinistra
13	op. condizionale	`?...:`	a destra
14	op. assegnamento e sue varianti	`=` `+=` `-=` `*=` `/=` `%=` `&=` `^=` `|=` `<<=` `>>=`	a destra
15 (min)	op. concatenazione	`,`	a sinistra

Esercizio 2

(espressioni e priorità degli operatori)

Scova l'errore…

Aldo, Giovanni e Giacomo hanno comprato in pasticceria una torta già tagliata in 12 fette, e poi si sono incontrati per mangiarla assieme.

E' andata più o meno così:

- Aldo ha mangiato 2 fette.
- Giovanni ne ha prese 5 nel piatto, ma poi si è ricordato di essere a dieta e quindi ne ha restituite 3.
- Giacomo ne ha prese 6, ma poi dopo averne mangiate 2 ha restituito le altre… salvo che di nascosto ne ha mangiato un' ulteriore fetta…

Hanno scritto quindi un programma che calcola il numero di fette rimaste nel piatto… al numero iniziale di fette hanno sottratto le fette mangiate da ogni membro della famiglia…

Esercizio 2

(espressioni e priorità degli operatori)

```
#include <stdio.h>
int main(void)
{
   int torta_i = 12;
   int torta_f;

   torta_f = torta_i - 2 - 5-3 - 6-4-1;
   return (0);
}
```

Il programma è **sbagliato**… perchè?

1. Correggete l'espressione inserendo le parentesi nei punti giusti, e senza sostituire all'operatore '-' l'operatore '+' …
2. Verificate tramite il debugger che il numero di fette finale calcolato sia effetivamente quello giusto…

Input e output in C

- Input con formato:

  scanf(“stringa formato”, lista variabili);

- Output con formato:

  printf(“stringa formato”, lista variabili);

  – Tramite la stringa di formato, si specifica “come”

  – Tramite i parametri successivi, si specifica “che cosa”

Esempio
(input output)

Esempio – Echo di un numero intero

- Realizzare (cioè scrivere e compilare) un programma che legga da tastiera un numero intero e ne stampi il valore a video (echo)
- pseudo-algoritmo:
  - Leggo da input un numero intero
  - Salvo il numero letto in una variabile apposita
  - Stampo a video il valore della variabile

Esempio

```
#include <stdio.h>
int main()
{
  int value;

  scanf("%d", &value);
  printf("Valore letto:%d\n", value);
  return 0;
}
```

Esercizio 1
(input/output)

Esercizio 1 – Echo di caratteri

- Realizzare un programma che legga da tastiera tre caratteri e ne stampi il valore a video (echo)

Esercizio 2

(IO, semplice programma)

- Realizzare un programma che legga da input tre numeri interi e stampi a video la loro somma e la media.

Esempio
(espressioni condizionali)

Esempio – Stabilire il massimo tra due valori

- Realizzare un programma che legga da input due numeri reali, e ne stampi a video il valore massimo
- Al fine di determinare il massimo, si utilizzino le sole espressioni condizionali

Esempio
(espressioni condizionali)

```
#include <stdio.h>
int main()
{
  float num1, num2, max;

  scanf("%f %f", &num1, &num2);

  max = ((num1 > num2) ? num1 : num2);
  printf("Max: %f\n", max);
  return 0;
}
```

Esercizio 1
(espressioni condizionali)

Elaborazione di numeri reali

- Realizzare un programma che legga da input un numero reale, e stampi a video:
  1. il suo valore assoluto
  2. il valore assoluto della sua parte intera

Esercizio 2
(espressioni condizionali)

Stampa di caratteri in ordine alfabetico

- Realizzare un programma che legga da input tre caratteri e li stampi in ordine alfabetico.

A tal scopo, si rammenti la rappresentazione dei caratteri in linguaggio C…

- Si utilizzino solo le espressioni condizionali (e non l'istruzione if… ad esempio)