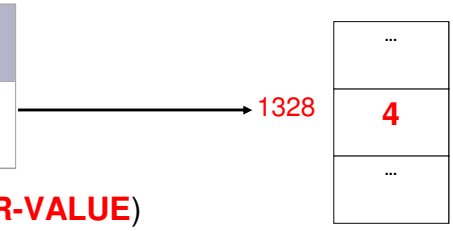
Tipi di dato primitivo e operatori

Variabili

- Una *variabile* è un'astrazione della cella di memoria
- Formalmente, una **variabile** è un **simbolo** associato a un *indirizzo fisico* (**L-VALUE**)

simbolo	*indirizzo*
x	**1328**



- che **denota un valore** (**R-VALUE**)
- perciò, l' R-VALUE di x è attualmente 4

Costanti (*numeriche*)

- Una ***costante*** è un'astrazione simbolica di un valore
  - Costante intera
    - In decimale: 12, 1234,…
    - in ottale (anteporre *0*): 014
    - In esadecimale (anteporre *0x*): 0x1f
  - Costante reale (floating point)
    - Con punto decimale: 24.0, 2.32
    - Con esponente: 2.4e1
    - **Attenzione!**
      24 è costante intera, 24.0 è costante floating point

Costanti e Variabili

- Una ***costante*** è un'astrazione simbolica di un valore
- L'associazione simbolo-valore non cambia mai durante l'esecuzione
- Una ***variabile*** è un **simbolo** associato a un *indirizzo fisico* (**L-VALUE**) che contiene un valore (**R-VALUE**)
- L'associazione simbolo-indirizzo non cambia mai durante l'esecuzione, ma può cambiare l'associazione indirizzo-valore
- Pertanto, nel caso di variabile, ad uno stesso simbolo possono corrispondere valori differenti in diversi momenti dell'esecuzione del programma

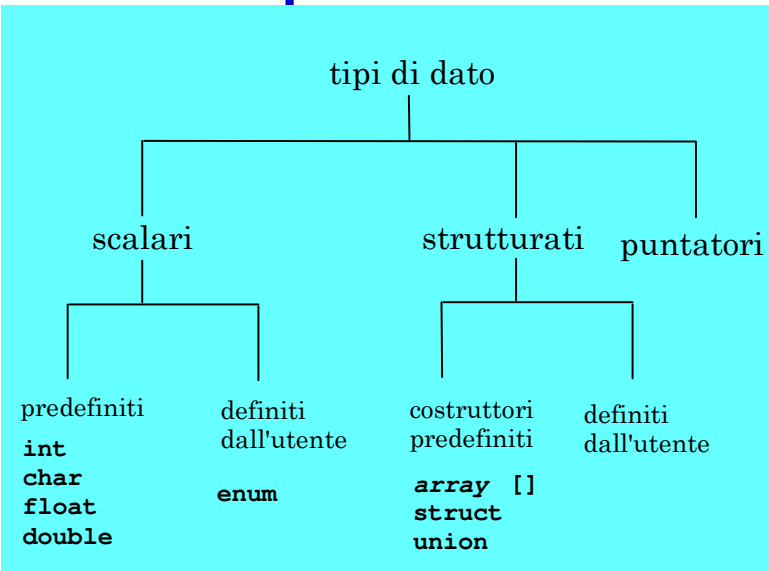
Attenzione
- **R-VALUE** può cambiare nel corso dell'esecuzione
- **L-VALUE** è fissato (e non cambia durante l'esecuzione)

Tipi di dato

- In C a ciascuna variabile è associato un **TIPO**
  - la classe di valori che la variabile può assumere nel corso dell'esecuzione del programma (e quindi gli operatori applicabili al valore in essa contenuto)
  - L'associazione di un nome ad un tipo di dato non cambia mai durante l'esecuzione del programma.
- In C a ciascuna costante è associato un **TIPO**

Quali sono i tipi di dato ammissibili in C?

Tipi di dato



Tipi di dato

- interi
- reali
- caratteri

- Per ciascun tipo consideriamo i seguenti aspetti:

  1. intervallo di definizione (se applicabile)
  2. notazione per le costanti
  3. operatori
  4. predicati (operatori di confronto)
  5. formati di ingresso/uscita

Tipo "int"

- Il tipo "int" è ben diverso dal tipo INTERO inteso in senso matematico dove

  INTERO **Z** $\{-\infty,\ldots,-2,-1,0,+1,+2,\ldots,+\infty\}$

- Infatti il tipo "int" ha un insieme di valori e di operazioni definibili su di essi <u>limitato a priori:</u>
  - *L'insieme dei valori dipende dalla macchina*
  - Normalmente il tipo "int" è memorizzato in una PAROLA DI MACCHINA (***WORD***) tipicamente lunga 16, 32 o 64 bit

bit, byte e word

BIT

- unità elementare dell'informazione trattata da un elaboratore
- unità di definizione di uno stato logico
  - rappresentazione logica del bit: {0, 1}.

BYTE

- la più piccola unità di memoria indirizzabile
- è composto da 8 bit

WORD

- dimensione nativa dei dati
- gruppo di bit che sono gestiti come unità dal computer

Leggere e visualizzare interi

- In generale,
  - la funzione per leggere un valore e assegnarlo ad una variabile è `scanf()`
  - La funzione per visualizzare il stringhe e/o il contenuto di una variabile è `printf()`
  - `printf()` e `scanf()` si trovano nella libreria `stdio.h` che deve quindi essere inclusa nei programmi che usano queste funzioni
- I formati per leggere interi è:
  - ✓ `%d` per variabili di tipo `int`

Esempio

- Scrivere un programma che legge due numeri da tastiera e ne stampa il risultato

```
#include <stdio.h>
main()
{
   int integer1;
   int integer2;
   int sum;

   scanf("%d",&integer1);
   scanf("%d",&integer2);
  sum = integer1 + integer2;
   printf("Risultato somma: %d\n",sum);
}
```

Inclusione della libreria

%d : il valore atteso è di tipo intero
& deve precede il nome della variabile

%d : il valore stampato è di tipo intero

I qualificatori del tipo `int`

4 qualificatori di tipo

short	`short [int] integer1;`
long	`long [int] integer1;`
signed	`[signed] int integer1;`
unsigned	`unsigned [int] integer1;`

- **SHORT** e **LONG** condizionano lo spazio allocato dal compilatore per la memorizzazione delle variabili del tipo definito
  (*Lo spazio effettivamente utilizzato dipende dalla macchina)*
  Formati di lettura per short e long (da usare all'interno della scanf()):
  - ✓ `%ld` per variabili di tipo `long(int)`
  - ✓ `%hd` per variabili di tipo `short(int)`
- **SIGNED** e **UNSIGNED** condizionano l'uso che si può fare della memoria allocata
  - I numeri unsigned sono sempre positivi
- In particolare, i qualificatori nelle loro combinazioni condizionano l' insieme dei valori assumibili da una variabile

Tipicamente…

Se una macchina ha una parola di 32 bit

SHORT INT	16 bit
INT	32 bit
LONG INT	32 bit

Se una macchina ha una parola di 16 bit

SHORT INT	16 bit
INT	16 bit
LONG INT	32 bit

Se una macchina ha una parola di 64 bit

SHORT INT	16/32 bit
INT	64 bit
LONG INT	64 bit

Gli **UNSIGNED INT** rappresentano valori interi non negativi e quindi guadagnano un bit per ampliare il range: $[0, 2^{16}-1]$ 16 bit
$[0, 2^{32}-1]$ 32 bit

Tipo "int" (*cont.*)

Per esempio se la macchina ha 32 bit un compilatore può (non è detto che ciò avvenga)

- prevedere un'allocazione di memoria inferiore per uno SHORT INT:
  16 bit *$[-2^{15}, 2^{15}-1]$ ovvero [-32768, +32767]*
- e superiore per un LONG INT:
  32 bit *$[-2^{31}, 2^{31}-1]$ ovvero [-2147483648, +2147483647]*

Ogni tentativo di assegnare un valore al di fuori del rango ammissibile, produce un errore di **OVERFLOW** o **UNDERFLOW**

a = b+c può dare un risultato sbagliato se la somma risulta essere:
>127 per i char, > di 32767 per gli int, e > 2147483647 per i longint.

Tuttavia, non si ha alcuna garanzia sulla dimensione di allocazione. E' possibile affermare con sicurezza solo che per ogni compilatore:

Spazio allocato (SHORT INT) <= Spazio allocato (INT) <=
<=Spazio allocato (LONG INT)

Ovvero **sizeof (SHORT INT) <= sizeof (INT) <= sizeof(LONG INT)**

Limiti

- I compilatori che seguono le raccomandazioni dello standard ANSI C mantengono all'interno dello header file <limits.h> l'indicazione del range di valori rappresentabili nei tipi interi.
- Ad esempio:

SHRT_MAX	32.767
SHRT_MIN	-32.768
USHRT_MAX	65.535
INT_MAX	32.767
INT_MIN	-32.768
UINT_MAX	65.535
LONG_MAX	2.147.483.647
LONG_MIN	-2.147.483.648
ULONG_MAX	4.294.967.295

Operatori aritmetici "int"

I seguenti operatori aritmetici sono quelli più comunemente applicati al tipo **int** (e tipi ottenuti da questo mediante qualificazione) :

Operatore aritmetico		Risultato
+	Addizione	int
-	Sottrazione	int
*	Moltiplicazione	int
/	Divisione intera	int
%	Modulo (resto dalla divisione intera)	int

What time is it?

Da secondi a minuti:secondi

- A quanto corrispondono 67 secondi?
  - 67%60 = 7 sec
  - 67/60 = 1 min
  - →1 min 7 sec

Da secondi a ore:minuti:secondi

- A quanto corrispondono 6007 secondi?
  - 6007%60 = 7 sec
  - 6007/60 = 100 min
  - 100%60 = 40 min
  - 100/60 = 1 h
  - → 1 h 40 min 7 sec

What time is it?

Dato un tempo espresso in secondi trasformarlo nel formato ore:minuti:secondi

```
#include <stdio.h>

void main(){

   int sec, min, h;

   printf("Digitare i secondi da convertire\n");
   scanf("%d", &sec);

   min = sec/60;
   sec = sec%60;
   h = min/60;
   min = min%60;

   printf("I secondi digitati corrispondono a %d h %d min %d sec", h, min, sec);

}
```

Esercizio

Scrivere un programma per il calcolo del resto per macchine per pagamenti automatici. Si supponga che la macchina disponga di un numero ***illimitato*** di monete da € 1 e € 2 e di banconote da € 5.

Il programma deve ricevere in input la cifra spesa e quella inserita dal cliente per il pagamento e stampare in output la composizione del resto formata dal ***minor numero di pezzi possibili***.

Operatori relazionali "int"

Operatore relazionale		Risultato
==	Operatore relazionale di uguaglianza (simbolo **=** denota l'operazione di assegnamento!)	(0 se falso, ≠ 0 se vero)
!=	Operatore relazionale di diversità	(0 se falso, ≠ 0 se vero)
>	Operatore relazionale di maggiore stretto	(0 se falso, ≠ 0 se vero)
<	Operatore relazionale di minore stretto	(0 se falso, ≠ 0 se vero)
>=	Operatore relazionale di maggiore-uguale	(0 se falso, ≠ 0 se vero)
<=	Operatore relazionale di minore-uguale	(0 se falso, ≠ 0 se vero)

Esercizio

Indicare se l'espressione è vera o falsa

Dati	Espressione	Vero/Falso
int a=3,b=2;	((b*b+1)%2) == (a-2)	
int a=-2,b=3;	((b*b)%2) != (a+3)	
int a=1, b=3;	((b-a%2)+2*a) > (a+b%2)	
int a=4, b=-1;	((a+b*3)%3) >= (a-(b*a))	

Esercizio

Indicare se l'espressione è vera o falsa

Dati	Espressione	Vero/Falso
int a=3,b=2;	((b*b+1)%2) == (a-2)	Vero
int a=-2,b=3;	((b*b)%2) != (a+3)	Falso
int a=1, b=3;	((b-a%2)+2*a) > (a+b%2)	Vero
int a=4, b=-1;	((a+b*3)%3) >= (a-(b*a))	Falso