Tipi di dati fondamentali

Linguaggi di Programmazione I

Ferdinando Cicalese

Tipi di dati fondamentali

- caratteri: *char* *signed char* *unsigned char*
- interi *short* *int* *long*
- interi *unsigned short* *unsigned* *unsigned long*
- reali *float* *double* *long double*

Utilità dei tipi di dati nelle dichiarazioni

- allocazione di spazio
- tipo di operazioni
  - Esempio: a + b

  A livello macchina l'operazione applicata a due variabili a e b di tipo *int* è diversa da quella applicata a due variabili a e b di tipo *float*

Il tipo di dati char

- In C si può utilizzare sia *int* che *char* per rappresentare i caratteri
  - costanti come 'b' '&' sono di tipo *int*
- Ogni *char* occupa 1 byte di memoria
  - Sulla maggior parte delle macchine 1 byte = 8 bit
- $2^8 = 256$ valori differenti (non tutti stampabili)
  - Esempi di caratteri stampabili: lettere, cifre, simboli di interpunzione, simboli speciali come & e %.
  - Esempi di caratteri non stampabili o di controllo: \n (newline) \t (TAB) \' (apice) \" (doppi apici)

Codice ASCII

- associa a ciascun carattere un valore intero

costante carattere	valore
'a'	97
'b'	98
.	.
'z'	112
'A'	65
'B'	66
.	.
'Z'	90
'}'	125
'1'	49

Codice ASCII

- nelle macchine che utilizzano il codice ASCII un carattere sara' memorizzato come un numero binario di 8 bit (byte)
  - 'A' è memorizzata come | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |

    valore ASCII di 'A' in base 2
- è possibile scrivere una costante carattere anche mediante il *carattere di escape* \ seguito dal valore ASCII del carattere scritto in notazione ottale

  Esempio:

  printf("%c", '\46'); stampa &

Esempio dell'uso dei caratteri

```
/* File: ex_caratteri.c
* Esempio uso caratteri
*/
#include <stdio.h>
main()
{
char x;
x = 'a';
printf ("%c ha valore ASCII %d\n", x, x);
x=x+5;
printf ("%c ha valore ASCII %d\n", x, x);
```

stampa :
a ha valore 97

stampa :
f ha valore 102

Il tipo di dati int

- Le variabili di tipo *int* memorizzano gli interi
- Un *int* occupa una parola della macchina
  - 2 byte o 4 byte (dipende dalla macchina)
  - Il numero di valori distinti memorizzabili varia a seconda della macchina
- L'intervallo di valori rappresentabili è $[-2^{wordsize-1}, 2^{wordsize-1}-1]$
  - Con 2 byte (16 bit) si va da -2^{15} a $+2^{15}-1$ ($2^{15}=32768$)
  - Con 4 byte (32 bit) si va da -2^{31} a $+2^{31}-1$ ($2^{31} \sim 2$ miliardi)
- Se un'espressione assume un valore intero che non ricade nell'intervallo degli interi rappresentabili allora si verifica un *integer overflow*

I tipi di dati short, long

- *short* è usato al posto di *int* in casi in cui la disponibilita' di memoria sia critica
  - il compilatore può allocare meno memoria per uno *short* che per un *int*
- di solito *short* occupa due byte
  - in una macchina con parole di 2 byte *int* e *short* hanno la stessa dimensione
- *long* è usato al posto di *int* in casi in cui si debbano usare interi grandi
  - il compilatore può allocare più memoria per uno *long* che per un *int*
- di solito *long* occupa 4 byte
- in una macchina con parole di 4 byte *int* e *long* hanno la stessa dimensione

Il tipo di dati unsigned

- Le variabili di tipo *unsigned* memorizzano gli interi privi di segno
- Un *unsigned* occupa una parola della macchina (come *int*)
- L'intervallo di valori rapresentabili è $[0, 2^{wordsize}-1]$
  - Con 2 byte (16 bit) si va da 0 a $+2^{16}-1$
  - Con 4 byte (32 bit) si va da 0 a $+2^{32}-1$
- Un *unsigned long* occupa lo stesso numero di byte di un l*ong*

Un *unsigned short* occupa lo stesso numero di byte di uno *short*

Costanti intere

- Una costante intera puo' essere seguita da un suffisso che ne specifichi il tipo
  - u oppure U per *unsigned* Esempio : 45U
  - l oppure L per *long* Esempio : 45L
  - ul oppure UL per *unsigned long* Esempio : 45UL
- Nel caso il suffisso non venga utilizzato allora il sistema sceglie tra *int, long* e *unsigned* il primo tipo in grado di rappresentare il valore della costante
  - esempio: con parole di 2 byte 32000 è di tipo *int* e 33000 è di tipo *long*

I tipi di dati reali: *float* e *double*

- Rappresentano i numeri reali
- Il compilatore può allocare più memoria per un *double* che per un *float*
  - su molte macchine un *float* occupa 4 byte
  - su molte macchine un *double* occupa 8 byte
- Il compilatore può allocare più memoria per un long *double* che per un *double*

I tipi di dati reali: *float* e *double*

- I possibili valori assegnabili ad una variabile di tipo reale sono descritti da
  - precisione: numero di cifre (decimali) significative
  - intervallo (range): minimo e massimo valore reale rappresentabili
- su molte macchine un *float* ha una precisione di circa 6 cifre significative e range pari approssimativamente a $[10^{-38},10^{+38}]$

  rappr: $0.d_1\,d_2\,d_3\,d_4\,d_5\,d_6 \times 10^n$, ogni d_i è decimale e $d_1 > 0$; $-38 \le n \le 38$
- su molte macchine un *double* ha una precisione di circa 15 cifre significative e range pari approssimativamente a $[10^{-308},10^{+308}]$

  rappr: $0.d_1\,d_{2\ldots}\,d_{14}\,d_5 \times 10^n$, ogni d_i è decimale e $d_1 > 0$; $-308 \le n \le 308$

Costanti reali

- Deve contenere o un punto decimale o la parte esponenziale o entrambe
  - Esempio: 245.2684e-3 equivale a 245.2684 x 10^3=245268.4
    - 245 è la parte intera,
    - 2684 è la parte decimale,
    - e-3 è la parte esponenziale
- Una costante reale può essere seguita da un suffisso che ne specifichi il tipo
  - f oppure F per *float* Esempio : 4.5F
  - l oppure L per *long double* Esempio : 4.5L
- Nel caso il suffisso non venga utilizzato allora le costanti sono di tipo *double*

Operatore *sizeof*

- *sizeof* è un operatore unario che restituisce la dimensione in bytes di
  - un tipo di dati
  - di un'espressione
  - di un array
  - diuna struttura
- ha la stessa priorita' e associativita' (da destra) degli altri operatori unari

```
/* File: dim.c
* Stampa dimensioni di tipi di dato
*/
#include <stdio.h>
main()
{
int a=3,b=7;
printf ("a+b ha dim. %d\n", sizeof(a+b));
printf ("char=%d\n", sizeof(char));
printf ("short=%d\n", sizeof(short));
printf ("int=%d\n", sizeof(int));
printf ("long=%d\n", sizeof(long));
printf ("float=%d\n", sizeof(float));
printf ("double=%d \n", sizeof(double));
}
```

Riassumendo

sizeof(*char*) = 1

sizeof(*short*) ≤ sizeof(*int*) ≤ sizeof(*long*);

sizeof(*signed*) = sizeof(*unsigned*) = sizeof(*int*);

sizeof(*float*) ≤ sizeof(*double*) ≤ sizeof(*long double*);

Conversione di tipo

- Ogni espressione aritmetica ha:
  - un valore
  - un tipo
- Ad esempio x+y
  - se x e y sono *int*, allora anche x+y è *int*
  - se x è float e y è *int*, allora x+y è *float*

Regole di conversione aritmetica-1

- se uno dei due operandi è *long double* allora l'altro viene convertito a *long double*
- altrimenti se uno dei due operandi è *double* allora l'altro viene convertito a *double*
- altrimenti se uno dei due operandi è *float* allora l'altro viene convertito a *float*
- altrimenti se uno dei due è *unsigned long* allora l'altro viene convertito a *unsigned long*
- altrimenti se uno è *long* e l'altro è *unsigned* , ci sono 2 casi:
  1. se *long* può rappresentare tutti i valori di un *unsigned* allora l'*unsigned* viene convertito a l*ong*,
  2. se *long* non può rappresentare tutti i valori di un *unsigned* allora entrambi gli operandi vengono convertiti a *unsigned long*

Regole di conversione aritmetica-2

– altrimenti se uno dei due operandi è *long* allora l'altro viene convertito a *long*

– altrimenti se uno dei due operandi è *unsigned* allora l'altro viene convertito a *unsigned*

Esempio:

char c; short s, int i; unsigned u; long l; unsigned long ul; float f

Espressione	Tipo
c - s/i	int
ul + f	float
u * 2.0 –1	double
c + 3.0	double
u/6 + 9	unsigned
u – l	???

Cast (conversioni esplicite)

• L'operatore di cast forza il compilatore ad utilizzare un tipo diverso per le espressioni

– Esempi:
```
int i;
.....
a = (double) i;
j = (long) ( 'A' + 1.0);
....
x = (float) ( (int) y + 1);
.....
(double) (x = 89);
```

• ha la stessa priorita' e associativita' (da destra) degli altri operatori unari.

– Esempio: (float) i * 5 è equivalente a (float i) * 5

Alcuni esempi

I tipi di dati: i numeri di Fibonacci

• I numeri di Fibonacci sono definiti ricorsivamente come:

$f_0 = 0$, $f_1 = 1$, $f_n = f_{n-1} + f_{n-2}$ per n = 3,4,…

• I numeri di Fibonacci crescono esponenzialmente

– f_{46} = 1836311903 (quasi 2 miliardi!)

• Vogliamo calcolare la sequenza dei numeri di Fibonacci fino al 46-esimo numero e i corrispondenti quozienti di Fibonacci definiti come $q_n = f_n / f_{n-1}$

• Nel programma i numeri di Fibonacci devono essere di tipo *long* e i quozienti di tipo *double*

I tipi di dati: i numeri di Fibonacci

```
#include <stdio.h>
#define LIMIT 46
main()
{
 long f_n=0, f_n1=1, n, temp;
 printf(%7s %19s %29s\n", " n", "N_Fibonacci" ,"Q_Fibonacci");
 printf("%7d %19d \n", 0,0) ;          /* stampa f_0 */
 printf("%7d %19d \n", 1,1) ;         /*stampa f_1 */
 for(n=2; n<=LIMIT; n++){
   temp = f_n1;
   f_n1 += f_n;
   f_n = temp;
   printf("%7ld %19ld %29.16f\n", n, f_n1, (double) f_n1/f_n);
 }                                          /*stampa f_n e f_n/f_(n-1) */
}
```

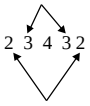
Uso degli array: sequenza palindroma

• una sequenza è palindroma se può essere letta sia da sinistra a destra che da destra a sinistra

• vogliamo scrivere un programma che prende in input una sequenza di al più 10 caratteri e verifica se la sequenza è palindroma

• per verificare se una sequenza è palindroma bisogna controllare che il primo carattere sia uguale all'ultimo, il secondo al penultimo, ecc….

• esempio:



Uso degli array: sequenza palindroma

```
#include <stdio.h>
#define TRUE 1
main()
{
int i = 0,j, palind = TRUE, lung;
 while ((a[i] =getchar() ) != '\n')
    i ++;
lung = i;   /* lunghezza della sequenza */
for (i = 0, j = lung -1; palind && i < lung/2 ; i++, j--)
   palind = (a[i] == a[ j ]);
if (palind)
 printf ("la sequenza è palindroma\n");
else
  printf("la sequenza non è palindromai\n");
}
```

Uso degli array: ordinamento

- vogliamo scrivere un programma che prende in input 10 numeri li memorizza in un array e li ordina in ordine crescente
- useremo un *algoritmo* chiamato *selection sort* :
  - cerca il più piccolo elemento dell'array e lo scambia con il primo elemento, poi cerca il secondo elemento dell'array e lo scambia con il secondo elemento e così via.
- Esempio: input 4213

  4213 ⟶ 1243 ⟶ 1243 ⟶ 1234

Uso degli array: ordinamento

```
#include<stdio.h>
main()
{
int i,j,min, temp, a[10];
for(i=0; i<=9; i++)
  scanf("%d", &a[i]);
for(i=0; i<=8; i++) {
  min = i;                      /* queste 4 linee*/
  for(j=i+1, j<= 9; j++)      /* cercano l'i-esimo */
    if(a[j]<a[min] )           /* elemento più */
      min=j;                   /* piccolo*/
  temp=a[min];
  a[min]=a[i]; /*scambia l'i-esimo elemento*/
  a[i]= temp; /* con l'i-esimo più piccolo*/
 }
for(i=0; i<=9; i++)
  printf("%d ", a[i]);
}
```

Esercizio:SORTING

```
/* File: insertion_sort.c
* Ordina  n  numeri usando insertSort
*/
...
for(i=1; i<= n; i++)
  {
        k=i-1;
        pivot = a[i];
        while(j > 0 && a[j] > pivot)
         {
                a[j+1] = a[j];
                j--;
         }
        a[j+1] = pivot;
}
```