Aritmetica dei puntatori

Marco Alberti

Programmazione e Laboratorio, A.A. 2016-2017

Dipartimento di Matematica e Informatica - Università di Ferrara
Ultima modifica: 7 dicembre 2016

sizeof

L'operatore `sizeof`, chiamato con il nome di un tipo `T` come argomento, restituisce la dimensione (in numero di byte) occupata in memoria dal tipo `T`.

Esempio

`printf("%d",sizeof(int))` stampa `4`.

L'operatore `sizeof` si può chiamare anche con il nome di una variabile come argomento: in tal caso restituisce la dimensione del tipo della variabile.

Esempio

Anche `int i; printf("%d",sizeof(i))` stampa `4`.

Invece `int i; printf("%d",sizeof((char)i))` stampa `1`.

Tipi primitivi

Esempio

- `sizeof(char)` = 1
- `sizeof(short)` = 2
- `sizeof(int)` = 4
- `sizeof(long)` = 8
- `sizeof(float)` = 4
- `sizeof(double)` = 8

Tipi composti

Date le seguenti dichiarazioni di tipo:

```
typedef struct { int x, y; } punto;

typedef char stringa[81];

typedef punto linea[10];

typedef struct { stringa nome; int eta; } persona;
```

- `sizeof(punto)` =

Tipi composti

Date le seguenti dichiarazioni di tipo:

```
typedef struct { int x, y; } punto;

typedef char stringa[81];

typedef punto linea[10];

typedef struct { stringa nome; int eta; } persona;
```

- `sizeof(punto)` = 8
- `sizeof(stringa)` =

Tipi composti

Date le seguenti dichiarazioni di tipo:

```
typedef struct { int x, y; } punto;

typedef char stringa[81];

typedef punto linea[10];

typedef struct { stringa nome; int eta; } persona;
```

- sizeof(punto) = 8
- sizeof(stringa) = 81
- sizeof(linea) =

Tipi composti

Date le seguenti dichiarazioni di tipo:

```
typedef struct { int x, y; } punto;

typedef char stringa[81];

typedef punto linea[10];

typedef struct { stringa nome; int eta; } persona;
```

- sizeof(punto) = 8
- sizeof(stringa) = 81
- sizeof(linea) = 80
- sizeof(persona) =

Tipi composti

Date le seguenti dichiarazioni di tipo:

```
typedef struct { int x, y; } punto;

typedef char stringa[81];

typedef punto linea[10];

typedef struct { stringa nome; int eta; } persona;
```

- `sizeof(punto)` = `8`
- `sizeof(stringa)` = `81`
- `sizeof(linea)` = `80`
- `sizeof(persona)` = `88`

Perché non `sizeof(persona)` = `85`? gcc *allinea* le strutture a multipli di 4 byte.

Puntatori e aritmetica

Il seguente programma

```
#include <stdio.h>
main() {
  int i = 100;
  int* p = (int*)100;
  printf("i = %d; i + 1 = %d\n", i, i + 1);
  printf("p = %d; p + 1 = %d\n", p, p + 1);
}
```

stampa

```
i = 100; i + 1 = 101
```

Puntatori e aritmetica

Il seguente programma

```
#include <stdio.h>
main() {
  int i = 100;
  int* p = (int*)100;
  printf("i = %d; i + 1 = %d\n", i, i + 1);
  printf("p = %d; p + 1 = %d\n", p, p + 1);
}
```

stampa

```
i = 100; i + 1 = 101
```

```
p = 100; p + 1 = 104
```

Aritmetica dei puntatori

- Il comportamento del programma precedente non è un errore: per motivi pratici, l'aritmetica dei puntatori è diversa da quella degli interi.
- Data la definizione `int *p;`, `p` contiene l'indirizzo di un intero; ad esempio, un elemento di un array di interi. Supponiamo che `sizeof(int)` sia `4`.
- Se l'aritmetica dei puntatori seguisse le regole di quella degli interi `*(p + 1)` sarebbe l'intero che inizia al secondo dei quattro byte dell'intero `*p` e si estende nel byte successivo, il che non ha significato.
- E' più intuitivo che (in aritmetica dei puntatori) `p + 1` punti al *successivo intero* in memoria, cioè al byte (in aritmetica intera) `p + sizeof(int)`, ossia `p + 4`.

Aritmetica dei puntatori

Se `T` è un tipo, data la definizione `T *p;` e se inizialmente `p` vale p_0:

Espressione	Indirizzo	Nuovo valore di `p`
`p + n`	$p_0 + \texttt{n} * \texttt{sizeof(T)}$	p_0
`p++`	p_0	$p_0 + \texttt{sizeof(T)}$
`++p`	$p_0 + \texttt{sizeof(T)}$	$p_0 + \texttt{sizeof(T)}$
`p += n`	$p_0 + \texttt{n} * \texttt{sizeof(T)}$	$p_0 + \texttt{n} * \texttt{sizeof(T)}$
`p - n`	$p_0 - \texttt{n} * \texttt{sizeof(T)}$	p_0
`p--`	p_0	$p_0 - \texttt{sizeof(T)}$
`--p`	$p_0 - \texttt{sizeof(T)}$	$p_0 - \texttt{sizeof(T)}$
`p -= n`	$p_0 - \texttt{n} * \texttt{sizeof(T)}$	$p_0 - \texttt{n} * \texttt{sizeof(T)}$

Array e aritmetica dei puntatori

Se `T` è un tipo, data la definizione `T v[10];`, le espressioni `v[3]` e `*(v+3)` sono equivalenti.

Esempio

Che cosa stampa questo programma?

```
#include <stdio.h>

int main() {
  int a[10], i;
  for (i = 0; i < 10; i++)
    *(a + i) = i;
  for (i = 0; i < 10; i++)
    printf("%d\n", *(a + i));
  return 0;
}
```

Array e aritmetica dei puntatori

Esempio

Che cosa stampa questo programma?

```c
#include <stdio.h>

int main() {
  int a[10], *p, i;
  for (i = 0, p = a; i < 10; i++, p++)
    *p = i;
  for (i = 0, p = a; i < 10; i++, p++)
    printf("%d\n", *p);
  return 0;
}
```

Stringhe e aritmetica dei puntatori

Esempio

Che cosa stampa questo programma?

```
#include <stdio.h>

void f(char* s) {
  while (*s)
    printf("%c", *s++);
}

int main() {
  f("Hello\n");
  return 0;
}
```

Stringhe e aritmetica dei puntatori

Esempio

Che cosa stampa questo programma?

```
#include <stdio.h>

void f(int* s) {
  while (*s)
    printf("%c", *s++);
}

int main() {
  f("Hello\n");
  return 0;
}
```