Array e puntatori - II

Laboratorio di Informatica I

Ferdinando Cicalese

Gli array come parametri formali di funzione

- L'indirizzo di base di un array puo` essere dichiarato come parametro formale di una funzione nei due modi seguenti:
  - utilizzando la notazione con parentesi quadrate degli array
  - come puntatore
- Esempio: int f(int [ ]); e` equivalente a int f(int *);

```
int f (int a[ ])   e` equivalente a   int f( int *a)
{                                     {
…..                                   …..
 }                                     }
```

Gli array come parametri formali di funzione

- Passando il nome di un array, si passa alla funzione l'indirizzo (chiamata per riferimento)
- In C non è possibile il passaggio di un array per valore
- Se si vuole che una funzione non modifichi l'array passato come parametro allora lo si deve copiare in un array temporaneo

Array come parametri di funzione: "InArray"

```
#define DIM 15
void InArray (int[ ], int);
main( )
{
int a[DIM];
printf ("Inserisci  %d interi \n", DIM);
InArray(a, DIM);
}

void InArray (int arr[ ], int n)
{int i;
if (n > 0)
for ( i = 0; i < n; i++)
 scanf("%d", &arr [ i ]);
}
```

- Prototipo di *InArray()*
- Chiamata di *InArray()*
- Definizione di *Inarray()*
- *Inarray()* legge n interi dalla schermo e li inserisce nell'array arr[ ]

Array come parametri di funzione: "InArray (versione con puntatore)"

```
#define DIM 15
void InArray (int *, int);
main( )
{
int a[DIM];
printf ("Inserisci  %d interi \n", DIM);
InArray(a, DIM);
}
void InArray(int *p, int n)
 {
  int i;
  for (i=0; i < n; i++)
    scanf ("%d", p++ );
 }
```

- Prototipo di *InArray()*
- Il primo parametro e` un puntatore
- Chiamata di *InArray()*
- Definizione di *Inarray()*
- *Inarray()* legge n interi dalla schermo e li inserisce nell'array arr[ ]

Array come parametri di funzione: "Sum"

```
#define DIM 15
void InArray (int[ ], int);
int sum (int[ ], int);
main( )
{int a[DIM];
InArray(a, DIM);
printf ("La somma è %d\n", sum(a, DIM));
}
int sum (int a[ ], int n)
{
int i, sum = 0;
for ( i = 0; i < n; i++)
sum += a [ i ];
return (sum);
}
```

- Prototipi di:
  - *InArray()*
  - *sum ()*
- Chiamata di *InArray*
- Chiamata di *sum*
- Definizione di *sum()*
- La funzione *sum()* restituisce la somma degli elementi a[0],a[1],…,a[n-1]

Array nelle chiamate di funzioni: "Sum (I versione)"

```
#define DIM 15
void InArray (int[ ], int);
int sum (int[ ], int);
main()
{
int a[DIM];
InArray(a, DIM);
printf ("Somma = %d\n", sum(a, DIM) );
printf ("Somma dei primi 5 = %d\n", sum(a, 5) );
printf ("Somma degli ultimi 2 =%d\n", sum(&a[DIM-2], 2) );
}
```

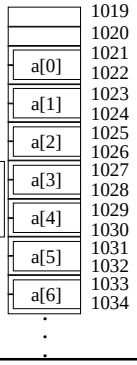
Comincia a sommare dal primo elemento

Comincia a sommare dal penultimo elemento

Funzionamento del passaggio di a[ ] come parametro di sum

• con la chiamata sum(a , 5) viene passato a *sum()* l'indirizzo di base di a[] **(= 1021)**

• la chiamata sum(&a[DIM-2] , DIM-2) passa a *sum()* l'indirizzo di a[DIM-2] **(= 1021 + (DIM - 2)* sizeof (int))**



Array nelle chiamate di funzioni: "Sum (II versione)"

```
void InArray (int[ ], int);
int sum (int * , int);
main()
{
int a[DIM];
InArray(a, DIM);
printf ("Somma = %d\n", sum (a, DIM) );
printf ("Somma primi 5 = %d\n", sump(a, 5) );
printf ("Somma degli ultimi 2 =%d\n",sum (&a[DIM-2], 2 ) );
}
int sum (int *p, int n)
{int i, sum = 0;
for ( i = 0; i < n; i++)
sum += * ( p + i ) ;
return (sum);
}
```

•Questa funzione *sum()* ha un puntatore come primo parametro

• p+i e` l'indirizzo dell' i-esimo elemento dell'array.

Array nelle chiamate di funzioni: "Sum (III versione)"

```
void InArray (int[ ], int);
int sum (int * , int);
main()
{
int a[DIM];
InArray(a, DIM);
printf ("Somma = %d\n", sum (a, DIM) );
printf ("Somma primi 5 = %d\n", sump(a, 5) );
printf ("Somma degli ultimi 2 =%d\n",sum (&a[DIM-2], 2 ) );
}
int sum (int *p, int n)
{int i, sum = 0;
for ( i = 0; i < n; i++, p++ )
sum += * p ;
return (sum);
}
```

• incrementa *p* ad ogni iterazione
• somma l'intero puntato da *p*

Allocazione dinamica di array

• Il compilatore effettua un'allocazione statica di un array in base alle dimensioni specificate nella dichiarazione

• Per allocare un array (ad es. di interi) dinamicamente:
  – si dichiara un puntatore (ad intero)
  – si alloca una zona di memoria di dimensioni adeguate
  – si usa come se fosse un array
  – al termine si libera la memoria

Le funzioni della libreria standard per l'allocazione

• *void * calloc (int n, unsigned int dim)*
  – alloca spazio contiguo in memoria per *n* elementi, ciascuno di dimensione *dim*
  – inizializza tutti i bit (in tale spazio di memoria) a zero
  – in caso di *successo* restituisce un puntatore di tipo ***void*** che punta all'indirizzo base; in caso di insuccesso restituisce un puntatore a ***null***

• Esempio:

```
int *p;
p = calloc( 10, sizeof (int) );
```

Le funzioni della libreria standard per l'allocazione

- *void * malloc (unsigned int dim_totale)*
  - alloca spazio in memoria per *dim_totale* bytes
  - non inizializza la memoria
  - in caso di successo restituisce un puntatore di tipo ***void*** che punta allo spazio richiesto; in caso di insuccesso restituisce un puntatore a ***null***
- Esempio:

```
int *p;
p = malloc( 10 * sizeof  (int) );
```

Le funzioni della libreria standard per l'allocazione

- *void free (void *ptr)*
  - se *ptr* non è *null* libera la zona di memoria puntata da *ptr*
  - se *ptr* è *null* non ha nessun effetto
  - provoca errore se la memoria puntata da *ptr* è stata già deallocata
- Esempio:

```
int *p;
p = calloc( 10, sizeof (int) );
free(p);
```

Esempio di allocazione dinamica

```
#include <stdlib.h>
int sum (int[ ], int);
void InArray (int *, int);
main()
{
int *a, num;
printf ("quanti elementi vuoi inserire?");
scanf ("%d",&num);
a = calloc ( num, sizeof(int) );
InArray (a, num);
printf ("%d", sum (a , num) );
free (a);
}
```

- Il file di intestazione *stdlib.h* contiene i prototipi di *calloc()* e *malloc()*
- Allocazione di *num* elementi di tipo *int* mediante ***calloc***
- Chiamata di *InArray*
- Viene liberata la memoria allocata

Esempio2: allocazione dinamica

```
#include <stdlib.h>
int sum (int[ ], int);
void InArray (int *, int);
main()
{
int *a, num;
printf ("quanti elementi vuoi inserire?");
scanf ("%d",&num);
a = (int *) malloc ( num * sizeof(int) );
InArray (a, num);
printf ("%d", sum (a , num) );
free (a);
}
```

- Il file di intestazione *stdlib.h* contiene i prototipi di *calloc()* e *malloc()*
- Allocazione(mediante ***malloc***)
  - cast a *(int *)*
  - *dim* interi
- Chiamata di *InArray*
- Viene liberata la memoria allocata

Allocazione dinamica

```
void InArray (int *, int);
void printarray (int *, int);
main()
{
int *a, num;
printf ("quanti elementi vuoi inserire?");
scanf ("%d",&num);
a = (int *) calloc ( num, sizeof(int) );
InArray (a, num);
printarray(p,num);
free (a);
}
void printarray (int *p, int size)
{ int i;
for (i=0; i < size; i++)
printf ("%d\n",  *(p++) );
}
```

- Chiamata di *printarray()*
- Definizione di *printarray()*
- Stampa l'intero puntato da *p* e poi incrementa il puntatore

Allocazione dinamica

```
void InArray (int *, int);
void printarray (int *, int);
main()
{
int *a, num;
printf ("quanti elementi vuoi inserire?");
scanf ("%d",&num);
a = (int *) calloc ( num, sizeof(int) );
InArray (a, num);
printarray(p,num);
free (a);
}
void printarray (int *p, int size)
{ int i;
for (i=0; i < size; i++)
printf ("%d\n",  *(p++) );
}
```

Riscrivere lo stesso programma utilizzando la funzione *malloc.*

Provare a:

allocare spazio per n valori

inserirne solo n-2

stampare gli n valori contenuti nello spazio allocato