Laboratorio di Algoritmi e Strutture Dati

Docente: Camillo Fiorentini

13 novembre 2007

Esercizio 1

a. Scrivere una funzione

```
void print(int a[], int n)
```

che stampa i primi n elementi dell'array `a[]` di `int` passato come primo parametro (quindi, deve stampare `a[0]`, ..., `a[n-1]`). Gli interi vanno separati da uno spazio e dopo l'ultimo intero va stampato `'\n'`.
Si ricordi che la notazione '`int a[]`' usata per un parametro di una funzione equivale a '`int *a`'.
Quindi:

- Il parametro `a` (di tipo `int*`) è l'indirizzo a un elemento dell'array.
- Il parametro `n` è il numero di elementi dell'array da stampare.

Che significato hanno le chiamate a `print()` nalla funzionione `mainA()`?

```
int mainA(){
  int v[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
  int *p = &v[3], *q = v+7;

  print(v, 10);
  print(&v[5], 3);
  print(v+5, 3);
  print(p+1, 2);
  print(&p[1], 2);
  print(q-5, 4);
  print(&q[-5], 4);
  return 0;
}
```

Verificare la risposta eseguendo il codice.

b. La chiamata

```
scanf("%d",  &n)
```

legge un intero (in notazione decimale) da standard input, saltando gli eventuali caratteri di spaziatura che lo precedono (spazi, "a capo", tabulazioni,...). Se la lettura ha successo, l'intero letto viene assegnato alla variabile `n` e la funzione `scanf()` restituisce 1. Se la lettura non ha successo la funzione restituisce 0 se il carattere letto non è convertibile in un intero, oppure `EOF` se si è letto "end-of-file".

Ad esempio, se lo standard input contiene

```
-123
```

a `n` viene assegnato l'intero -123 e la funzione restituisce 1. Se lo standard input contiene

```
a
```

la chiamata restituisce 0 e il carattere `a` non è letto.

Scrivere una funzione

```
int read(int a[])
```

che legge da standard input una sequenza di interi separati da uno o più spazi e memorizza i valori letti nell'array `a[]` passato come parametro. La lettura dell'input termina quando viene letto un carattere non convertibile in intero e la funzione restituisce il numero di interi letti.
Si *assume* che l'array passato come parametro sia sufficientemente grande per contenere i valori letti.

Esempio
Supponiamo che di eseguire la chiamata

```
read(v)
```

e che lo standard input contenga

```
10   -20   +30  q
```

La lettura termina quando viene letto `q` e il valore restituito è 3. La funzione pone:

```
v[0] = 10
v[1] = -20
v[2] = 30
```

Provare ad eseguire:

```
int mainB(){
  int v[100], n;
  n = read(v);
  print(v,n);
  if(n>2){
    print(v,n-2);
    print(v+2, n-2);
  }
  return 0;
}
```

c. Nel punto precedente, se vengono letti più di 100 interi il comportamento del programma è scorretto.
Per evitare di leggere più interi di quanti possa contenerne l'array, modificare la funzione `read()` in modo che legga al massimo `N` interi, dove `N` è una macro definita nel programma.

Provare ad eseguire:

```
int mainC(){
  int v[N], n;
  n = read(v);
  print(v,n);
   if(n>2){
    print(v,n-2);
    print(v+2, n-2);
  }
  return 0;
}
```

d. Scrivere una funzione `media()` per calcolare la media aritmetica degli elementi di un array di interi (definire in modo opportuno i parametri e il tipo della funzione).

e. Scrivere una funzione `inv()` il cui compito è quello di invertire gli elementi di un vettore passato come primo argomento, dalla posizione i (secondo argomento) alla posizione j (terzo argomento), dove si assume che $i < j$. Ad esempio, chiamando `inv()` passando come argomenti `a`, 50 e 100, al termine dell'escuzione `a[50]` contiene il valore che c'era in `a[100]` prima della chiamata, `a[51]` contiene il valore che c'era in `a[99]`, ..., `a[100]` contiene il valore che c'era in `a[50]`. Per effettuare lo scambio fra due elementi dell'array, usare la funzione `swap()`.

Provare a eseguire

```
int mainE(){
  int v[N], n;
  n = read(v);
  print(v,n);
  inv(v,0, n-1),
  print(v,n);
  inv(v, 1,  n-2);
  print(v,n);
  return 0;
}
```

Esercizio 2

Si vuole leggere una sequenza di interi terminante con un carattere non convertibile in intero (come nell'esercizio precedente) e stampare la sequenza in ordine inverso. Una soluzione è:

```
#define N ...

int main(){
  int v[N],n,k;
  n = read(v);
// la sequenza letta e' v[0], v[1], ... , v[n-1]
  for(k=n-1; k>=0; k--)
    printf("&d ", v[k]);
  putchar('\n');
  return 0;
}
```

Il programma è corretto solo per sequenze contenenti al più `N` interi.

Per evitare di usare gli array (che impongono un limite alla lunghezza della sequenza di input), si può definire una funzione *ricorsiva*

```
void printInv()
```

che stampa in ordine inverso la sequenza di interi letta in input (la sequenza termina con un carattere non convertibile in intero o con "end-of-file"). Ad esempio, se l'input è

```
1 2 3 a
```

la chiamata

```
printInv();
```

deve stampare in output

```
3 2 1
```

Ragionare per induzione sulla lunghezza della sequenza di input.

Esercizio 3

a. Scrivere una funzione `copy()` che copia nel vettore di interi `to[]` passato come primo parametro i primi n elementi del vettore di interi `from[]` passato come secondo parametro; n è passato come terzo parametro. Ad esempio, dopo la chiamata

```
copy(a, b, 50)
```

`a[0]` contiene il valore in `b[0]`, `a[1]` contiene il valore in `b[1]`,..., `a[49]` contiene il valore in `b[49]`.

Assumere che il vettore passato come primo parametro contenga spazio sufficiente per copiare n elementi.

Verificare cosa viene stampato dala funzione `mainA()` in `es3.c`.

b. Dato l'array

```
int z[8] = {0, 1, 2, 3, 4, 5, 6, 7};    // z[0]= 0, z[1]=1, ..., z[7]=7
```

porre tutti gli elementi di `z[]` uguali a 5 facendo *esclusivamente* tre chiamate alla funzione `copy()`.

Esercizio 4

Eseguire il programma `es4.c` e motivare l'output ottenuto. In compilazione, vengono segnalati dei warning. Come è possibile eliminarli? Dopo averli eliminati, cambia l'output?

È possibile che il programma si comporti in maniera diversa se eseguito su un sistema diverso da quello che si sta usando?